

Homework Week 7

MAEER's MIT

Q1. $J(\underline{w}) = \frac{1}{N} \|\underline{X}\underline{w} - \underline{b}\|_2^2 + \lambda \|\underline{w}\|_2^2 \quad \lambda > 0.$

a) $\nabla_{\underline{w}} J(\underline{w}) = \frac{2}{N} \underline{X}^T (\underline{X}\underline{w} - \underline{b}) + 2\lambda \underline{w}$

b) $\nabla_{\underline{w}} J(\hat{\underline{w}}) = 0$

$$\therefore \frac{2}{N} \underline{X}^T (\underline{X}\hat{\underline{w}} - \underline{b}) + 2\lambda \hat{\underline{w}} = 0.$$

$$\therefore \underline{X}^T (\underline{X}\hat{\underline{w}} - \underline{b}) + N\lambda \hat{\underline{w}} = 0.$$

$$\therefore (\underline{X}^T \underline{X} \hat{\underline{w}} + N\lambda \underline{I} \hat{\underline{w}}) = \underline{X}^T \underline{b} \quad \underline{I} \text{ is identity vector}$$

$$\therefore \hat{\underline{w}} = (\underline{X}^T \underline{X} + N\lambda \underline{I})^{-1} \underline{X}^T \underline{b}$$

Since The pseudoinverse solution is $\underline{w} = (\underline{X}^T \underline{X})^{-1} \underline{X}^T \underline{b}$

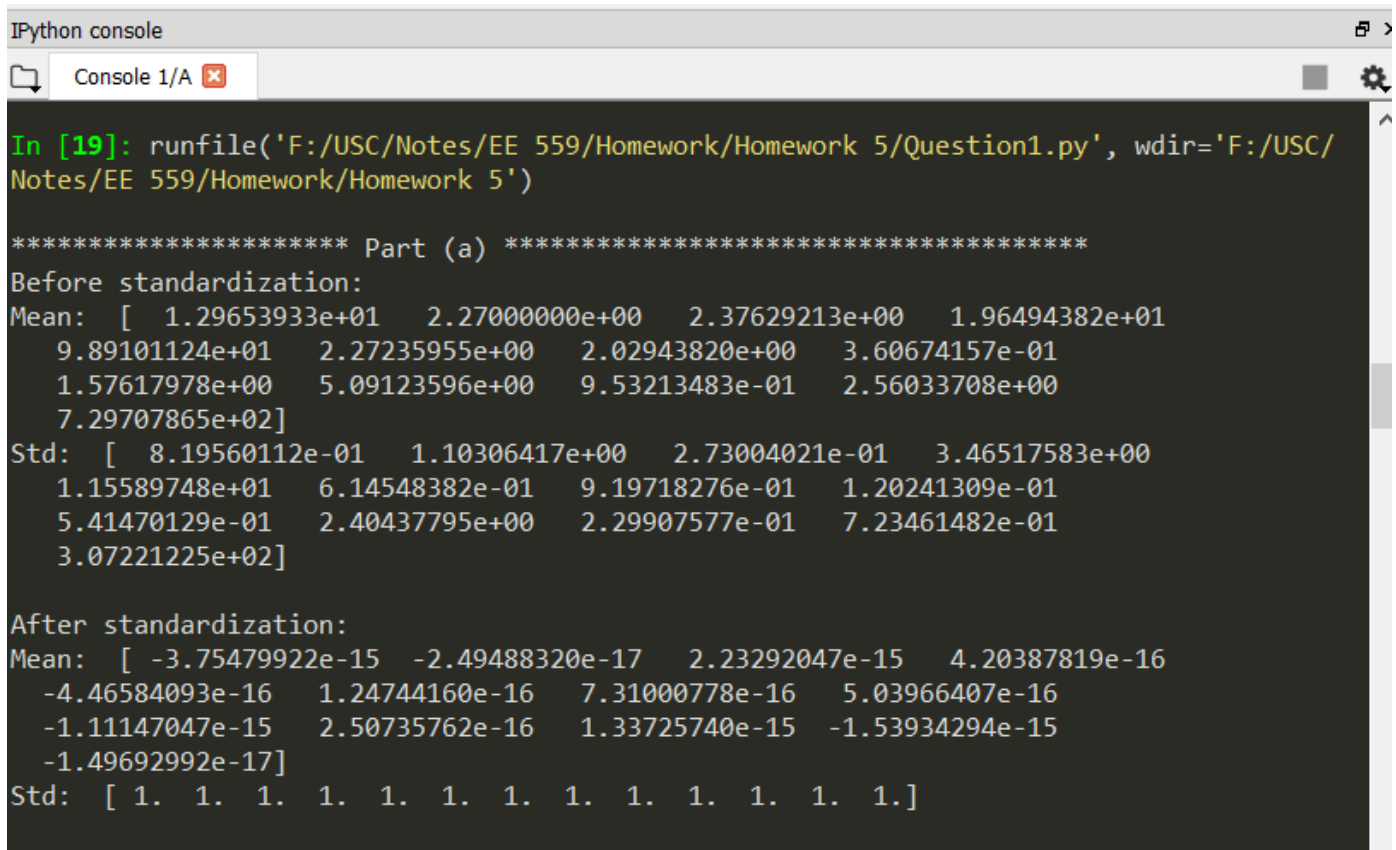
Since $N\lambda \underline{I}$ is non-singular, $(\underline{X}^T \underline{X} + N\lambda \underline{I})^{-1}$ will always exist.

EE 559 Homework Week 7

Question

2(a). I am using scikit-learn and python

2(b).



```
IPython console
Console 1/A x

In [19]: runfile('F:/USC/Notes/EE 559/Homework/Homework 5/Question1.py', wdir='F:/USC/Notes/EE 559/Homework/Homework 5')

***** Part (a) *****
Before standardization:
Mean: [ 1.29653933e+01  2.27000000e+00  2.37629213e+00  1.96494382e+01
 9.89101124e+01  2.27235955e+00  2.02943820e+00  3.60674157e-01
 1.57617978e+00  5.09123596e+00  9.53213483e-01  2.56033708e+00
 7.29707865e+02]
Std: [ 8.19560112e-01  1.10306417e+00  2.73004021e-01  3.46517583e+00
 1.15589748e+01  6.14548382e-01  9.19718276e-01  1.20241309e-01
 5.41470129e-01  2.40437795e+00  2.29907577e-01  7.23461482e-01
 3.07221225e+02]

After standardization:
Mean: [ -3.75479922e-15 -2.49488320e-17  2.23292047e-15  4.20387819e-16
 -4.46584093e-16  1.24744160e-16  7.31000778e-16  5.03966407e-16
 -1.11147047e-15  2.50735762e-16  1.33725740e-15 -1.53934294e-15
 -1.49692992e-17]
Std: [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

Figure 1. Mean and Standard Deviation before and after standardization

The normalizing factors should be calculated from the training data, because in real life classification problems, we are never given the testing data beforehand. And, even if we are provided with the testing data beforehand, we cannot peek into the testing data. Therefore, we cannot possibly know the normalizing factors of the testing data. So, we use the normalizing factors of the training data and apply it to the testing data.

2(c).

- (i) The default initial weight vector is **all zeros vector**. This can be seen in the source code of *stochastic_gradient.py* program in the scikit library, under the *_allocate_parameter_mem* function.
- (ii) There are two halting conditions. The first is *tol* argument in the *Perceptron model*. *Tol* argument defaults to none. But if set, the iterations will stop when the difference between weight vectors from two successive iteration is less than *tol*.
The second halting condition is the *max_iter* argument which is the maximum number of iterations, after which the weights are not updated.

2(d).

```
IPython console
Console 1/A x

***** Part (d) *****

Using Perceptron
Using 2 features:
Training Accuracy:  78.6516853933 %
Testing Accuracy:   75.2808988764 %

Weight vectors:
[[ 1.92786347 -2.73782803]
 [-2.83640786 -1.45957057]
 [ 0.71427739  2.32080787]]

Intercept vector:
[ 0. -1. -1.]

Using 13 features:
Training Accuracy:  100.0 %
Testing Accuracy:   94.3820224719 %

Weight vectors:
[[ 4.86476371  1.15133828  3.30200511 -1.22729962  1.8644008  -0.81068027
  3.56325529 -1.46895617  0.8142626  -1.8565385  -0.4955582  3.09855528
  4.76705126]
 [-4.79991664 -4.34244909 -3.04436393  2.2071938  -3.57035669  2.14316511
  0.68315986  3.50979604 -0.55674479 -4.84794504  1.39029976  0.69251957
 -5.9952772 ]
 [ 1.16203746  2.20295434  1.71335502  0.95135987  0.81847001  0.23457438
 -2.90672792 -3.37616962 -2.5724805   4.36595908 -3.83075284 -2.45759805
 1.84638021]]

Intercept vector:
[-5. -3. -6.]
```

Figure 2. Weight vectors and classification accuracy for training and testing dataset (Using 2 features and 13 features)

2(e).

```
IPython console
Console 1/A x

***** Part (e) *****

Using Perceptron with random initial weights
Using 2 features:
Maximum Training Accuracy: 83.1460674157 %
Corresponding Testing Accuracy: 78.6516853933 %

Weight vectors:
[[ 1.97416703 -1.88271945]
 [-1.7997393 -0.53322065]
 [-0.1446256  2.36207303]]

Intercept vector:
[-2.22769573 -1.00210834 -0.94817408]

Using 13 features:
Maximum Training Accuracy: 100.0 %
Corresponding Testing Accuracy: 93.2584269663 %

Weight vectors:
[[ 4.41668324  1.11438584  3.63333384 -3.22940429  1.14600102 -1.07430019
  2.9467539  -1.63127802 -2.78171984 -0.57314434  1.93088577  4.25997478
  4.80157113]
 [-6.44442293 -4.24476379 -4.55894182  0.89377655 -3.04617913  2.60894514
  0.28930763  3.02707746  1.06208913 -6.13791427  3.10030003 -1.29513616
 -7.38443912]
 [ 0.65408639  2.28864514  1.34304005  0.13291721  1.59825945  0.33868873
 -1.24562986 -1.81332425 -1.08076543  4.73297639 -0.7555974  -2.16946905
 -2.51744271]]

Intercept vector:
[-5.49263577 -5.60561185 -5.13412491]
```

Figure 3 Best performing weight vectors in 100 runs and corresponding classification accuracy (Using 2 features and 13 features)

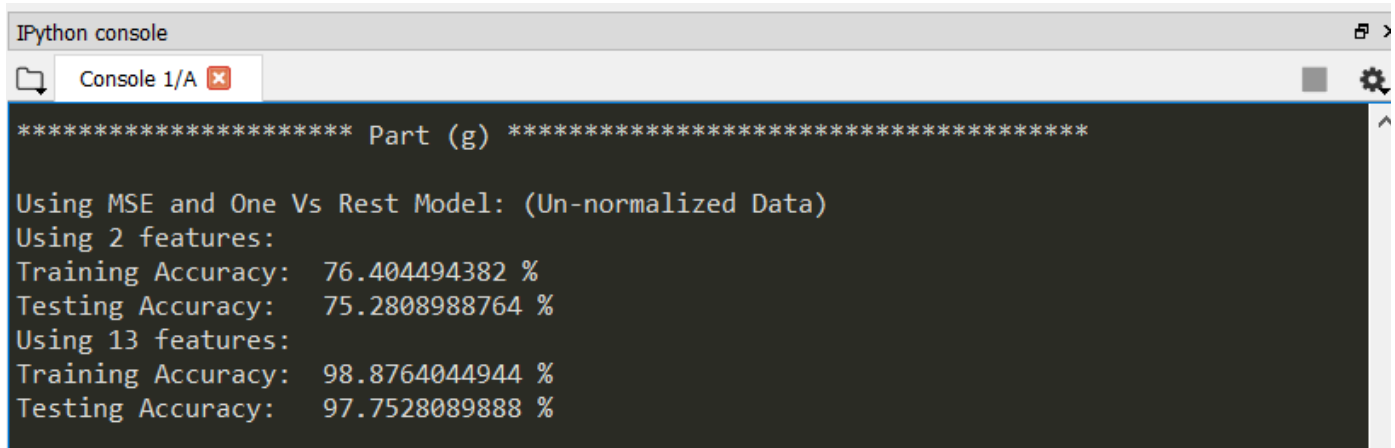
Firstly, we can observe that the testing and training accuracy for classification using 2 features is less than the accuracy for classification using 13 features (for both part (d) and part (e)). The reason for this is that the dataset is not completely linearly separable in the feature space comprising of only the first two features. On the other hand, the data is completely linearly separable in the feature space comprising of all the 13 features.

We can observe that for classification using 2 features, the accuracy rate in part (d) is less than that in part (e). I believe that the main reason behind this improvement of accuracy is the random

initialization of weight vectors. In part (d) we never initialized the weight vector, and the perceptron starts from a pre-determined weight vector and as it tries to minimize the gradient of criterion function, it gets stuck at some local minima, or overshoot the global minima, resulting in a lower accuracy rate. Thus, the final weights might not be the

However, in part (e), using random initial weight vectors in each of the 100 runs, the perceptron model may reach the global minima of the gradient of the criterion function.

2(g).



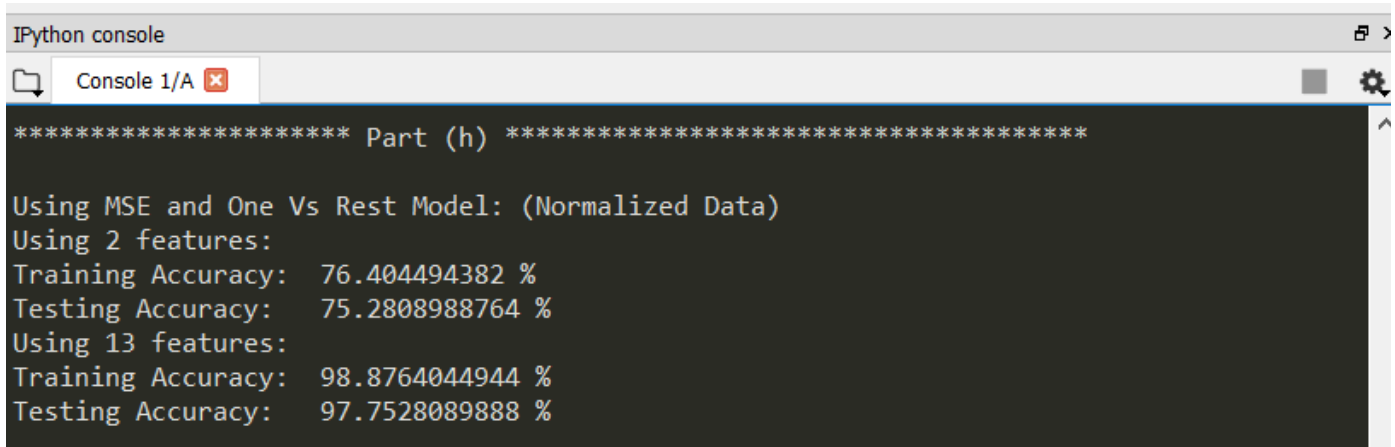
```
IPython console
Console 1/A x

***** Part (g) *****

Using MSE and One Vs Rest Model: (Un-normalized Data)
Using 2 features:
Training Accuracy: 76.404494382 %
Testing Accuracy: 75.2808988764 %
Using 13 features:
Training Accuracy: 98.8764044944 %
Testing Accuracy: 97.7528089888 %
```

Figure 4. Training and testing accuracy using the un-normalized data

2(h).



```
IPython console
Console 1/A x

***** Part (h) *****

Using MSE and One Vs Rest Model: (Normalized Data)
Using 2 features:
Training Accuracy: 76.404494382 %
Testing Accuracy: 75.2808988764 %
Using 13 features:
Training Accuracy: 98.8764044944 %
Testing Accuracy: 97.7528089888 %
```

Figure 5. Training and testing accuracy using the normalized data

2(i)

Intuitively, one would expect that the classification accuracy might be different for the normalized and the un-normalized data. But as seen in figure 4 and figure 5, the classification accuracies are the same in both the cases. This can be mathematically proven as follows:

Consider a dataset with just feature. In Augmented space it can be represented as:

$$X = \begin{bmatrix} 1 & 2 \\ 1 & 5 \\ 1 & 4 \end{bmatrix}$$

If we are simply standardizing the dataset (i.e. the data represented in your second column) we can represent this as a linear transformation A defined by the matrix:

$$A = \begin{bmatrix} 1 & \frac{-\mu_x}{\sigma_x} \\ 0 & \frac{1}{\sigma_x} \end{bmatrix}$$

Multiplying X by A (i.e. computing XA) is basically equivalent to subtracting the mean for each non-constant column and dividing by the standard deviation.

The least squares estimator is:

$$w = (X^T X)^{-1} X^T b$$

The OLS estimator on the transformed data XA is:

$$\begin{aligned} \hat{w} &= (A^T X^T X A)^{-1} A^T X^T b \\ \hat{w} &= A^{-1} b \end{aligned}$$

Thus, estimated coefficients are different, but the predictions are the same:

$$\hat{X} \hat{w} = (XA)(A^{-1}b) = Xw$$

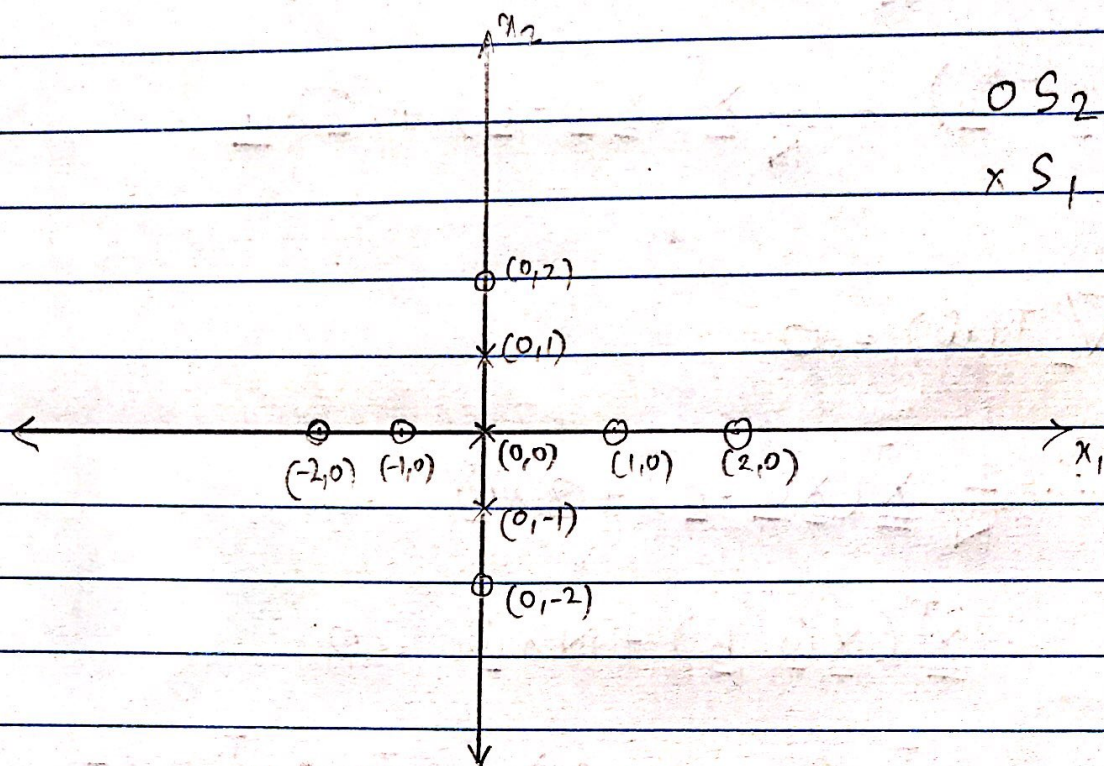
Therefore, the new criterion function $\hat{X} \hat{w}$ is equal to the old criterion function Xw . Thus, the classification accuracy will remain same for both normalized data and un-normalized data.

2(j).

In general, the classification accuracy for OLS should be greater than the classification accuracy for gradient descent approach. But in our case, we observe that for part (h) i.e. perceptron with gradient descent, the classification accuracy is greater than part (e) i.e. OLS. I think this can be attributed to the outlier data samples. Linear regression (OLS) tries to accommodate even the outlier data to find the best fitting curve, thus decreasing the accuracy. However, gradient descent is not affected due to such outlier data samples.

Q3. $S_1: (0,0)^T, (0,1)^T, (0,-1)^T$
 $S_2: (-2,0)^T, (-1,0)^T, (0,2)^T, (0,-2)^T, (1,0)^T, (2,0)^T$

a)



As we can observe, the data samples are not linearly separable.

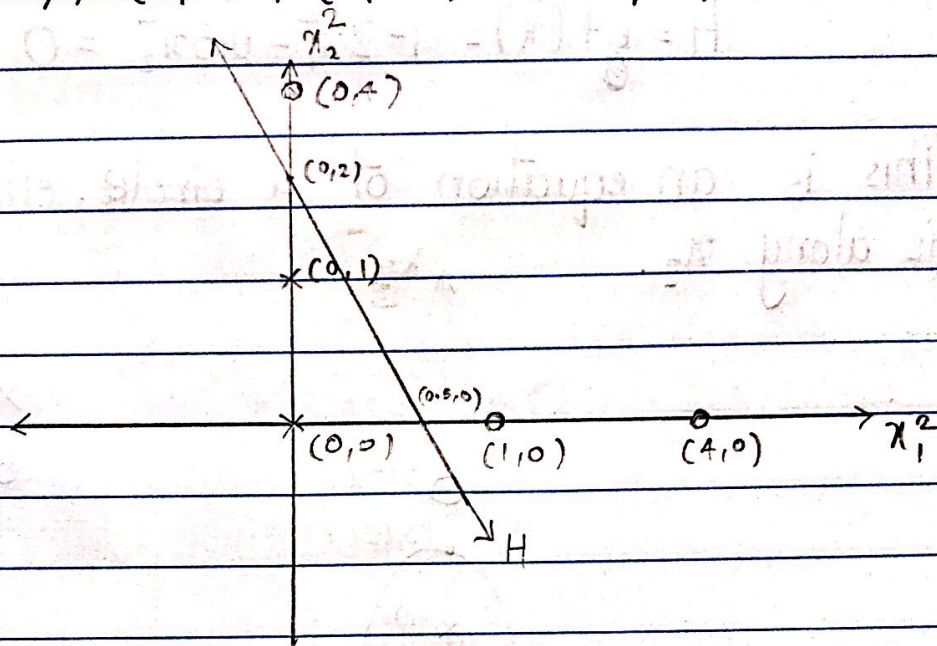
b) In the expanded feature space, the data points will be

1	$S_1:$	$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$	$S_2:$	$\begin{bmatrix} 1 \\ -2 \\ 0 \\ 4 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \\ 4 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ -2 \\ 0 \\ 4 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 2 \\ 0 \\ 4 \\ 0 \end{bmatrix}$
x_1		0	0	0		-2	-1	0	0	1	2
x_2		0	1	-1		0	0	2	-2	0	0
x_1^2		0	0	0		4	1	0	0	1	4
$x_1 x_2$		0	0	0		0	0	0	0	0	0
x_2^2		0	1	1		0	0	4	4	0	0

c) Considering only the (x_1^2, x_2^2) space, we get

$$S_1: (0,0)^T, (0,1)^T, (0,1)^T$$

$$S_2: (4,0)^T, (4,0)^T, (0,4)^T, (0,4)^T, (1,0)^T, (4,0)^T$$



The decision region for this space is given by:

$$H' = g(\underline{u}) = 1 - 2u_1 - 0.5u_2 = 0 \quad \begin{matrix} u_1 = x_1^2 \\ u_2 = x_2^2 \end{matrix}$$

$$= \underline{w}^T \underline{u}$$

$$\underline{w}^T = \begin{bmatrix} 1 \\ -2 \\ -0.5 \end{bmatrix} \quad \underline{u} = \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

Therefore, the complete weight vector is given by

$$\underline{w}' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ -0.5 \end{bmatrix}$$

$\begin{matrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{matrix}$

d) Therefore, the decision boundary in the original feature space will be

$$H = f(x) = 1 - 2x_1^2 - 0.5x_2^2 = 0$$

This is an equation of a ~~circle~~ ellipse with major axis along x_2 .

