Vectorized Logistic Classification

Vectorized Implementation of
Logistic Regression

Vectorized Implementation

- Recall, for the non-vectorized implementation of Logistic Regression, the batch update rule is:

For each feature ($j = 0;\ j <= n;\ j++$) {

$$\theta_j = \theta_j - \propto \frac{\partial}{\partial \theta_j} J(\theta)$$
$$= \theta_j - \propto slope_j$$

}

$slope_j$ for each $j = \{0, 1 \ldots n\}$ $= \frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^i \right) x_j^{(i)} = \frac{1}{m} \sum_{i=1}^{m} \left(g\left(z^{(i)}\right) - y^i \right) x_j^{(i)}$

$$h_\theta\left(x^{(i)}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}}} = \frac{1}{1 + e^{-z^{(i)}}} = g\left(z^{(i)}\right)$$

$$z^{(i)} = \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots \theta_n x_n^{(i)}, \qquad x_0^{(i)}=1$$

Vectorized Implementation

- Notice that over the sum of training examples, $z^{(i)}$ must be computed for each training example $i = \{1..m\}$:

$$slope_j = \frac{1}{m}\sum_{i=1}^{m}\left(g(z^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

- Thus, the summation operation over m training examples will produce m terms for $z^{(i)}$, which can be written as an m component vector $\boldsymbol{z} \in \mathbb{R}^m$:

$$\boldsymbol{z} = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$= \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \dots \\ z^{(m)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}^T \boldsymbol{x}^{(1)} \\ \boldsymbol{\theta}^T \boldsymbol{x}^{(2)} \\ \dots \\ \boldsymbol{\theta}^T \boldsymbol{x}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \cdots \theta_n x_n^{(1)} \\ \theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \cdots \theta_n x_n^{(2)} \\ \dots \\ \theta_0 x_0^{(m)} + \theta_1 x_1^{(m)} + \cdots \theta_n x_n^{(m)} \end{bmatrix}$$

Vectorized Implementation

- Moreover, a vector $g(\boldsymbol{z})$ is created:

$$g\left(z^{(i)}\right) = \frac{1}{1 + e^{-z^{(i)}}}$$

$$g(\boldsymbol{z}) = \begin{bmatrix} g\left(z^{(1)}\right) \\ g\left(z^{(2)}\right) \\ \dots \\ g\left(z^{(m)}\right) \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1 + e^{-z^{(1)}}} \\ \dfrac{1}{1 + e^{-z^{(2)}}} \\ \dots \\ \dfrac{1}{1 + e^{-z^{(m)}}} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1 + e^{-\left(\theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \cdots \theta_n x_n^{(1)}\right)}} \\ \dfrac{1}{1 + e^{-\left(\theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \cdots \theta_n x_n^{(2)}\right)}} \\ \dots \\ \dfrac{1}{1 + e^{-\left(\theta_0 x_0^{(m)} + \theta_1 x_1^{(m)} + \cdots \theta_n x_n^{(m)}\right)}} \end{bmatrix}$$

- The dimension: $g(\boldsymbol{z}) \in \mathbb{R}^{mx1}$

Vectorized Implementation

- Furthermore, $y^{(i)}$ is subtracted from the corresponding component of $g(\boldsymbol{z})$:

$$\begin{bmatrix} g(z^{(1)}) - y^{(1)} \\ g(z^{(2)}) - y^{(2)} \\ \dots \\ g(z^{(m)}) - y^{(m)} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-z^{(1)}}} - y^{(1)} \\ \frac{1}{1+e^{-z^{(2)}}} - y^{(2)} \\ \dots \\ \frac{1}{1+e^{-z^{(m)}}} - y^{(m)} \end{bmatrix}$$

- The dimension of the vector $\boldsymbol{y} \in \mathbb{R}^{mx1}$

Vectorized Implementation

- Finally, $x_j^{(i)}$ multiplies each difference component $\left(g(z^{(i)}) - y^{(i)}\right)x_j^{(i)}$:

$$\begin{bmatrix} \left(g(z^{(1)}) - y^{(1)}\right)x_j^{(1)} \\ \left(g(z^{(2)}) - y^{(2)}\right)x_j^{(2)} \\ \dots \\ \left(g(z^{(m)}) - y^{(m)}\right)x_j^{(m)} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{1+e^{-z^{(1)}}} - y^{(1)}\right)x_j^{(1)} \\ \left(\frac{1}{1+e^{-z^{(2)}}} - y^{(2)}\right)x_j^{(2)} \\ \dots \\ \left(\frac{1}{1+e^{-z^{(m)}}} - y^{(m)}\right)x_j^{(m)} \end{bmatrix}$$

- This can be viewed as a component by component product of vectors $(g(\boldsymbol{z})) - \boldsymbol{y})$ and $\boldsymbol{x_j}$

- The vector $(g(\boldsymbol{z})) - \boldsymbol{y})$ has m components.

- The vector $\boldsymbol{x_j}$ has m components.

Vectorized Implementation

- Therefore,

$$slope_j = \frac{1}{m}\sum_{i=1}^{m}\left(g\left(z^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

$$= \frac{1}{m} SumofRows \begin{bmatrix} \left(\frac{1}{1+e^{-z^{(1)}}} - y^{(1)}\right) x_j^{(1)} \\ \left(\frac{1}{1+e^{-z^{(2)}}} - y^{(2)}\right) x_j^{(2)} \\ \dots \\ \left(\frac{1}{1+e^{-z^{(m)}}} - y^{(m)}\right) x_j^{(m)} \end{bmatrix}$$

$$= \frac{1}{m}(g(\boldsymbol{z}) - \boldsymbol{y})\boldsymbol{x}_j$$

$x_j^{(i)}$: Training example i of feature j.

Vector product.

Vectorized Implementation

- Therefore to compute the $slope_j$,

$$slope_j = \frac{1}{m}(g(\boldsymbol{z}) - \boldsymbol{y})\boldsymbol{x}_j$$

1. Compute vector $g(\boldsymbol{z})$.
2. Subtract given vector $\boldsymbol{y}$ from $g(\boldsymbol{z})$.
3. Take the sum of the products obtained from the component wise multiplication of the vectors $(g(\boldsymbol{z}) - \boldsymbol{y})$ and $\boldsymbol{x}_j$.

Dimensionality Analysis

- Consider the dimension of the vectors in $slope_j$:

$$slope_j = \frac{1}{m}(g(\boldsymbol{z}) - \boldsymbol{y})\boldsymbol{x}_j$$

- $g(\boldsymbol{z}) - \boldsymbol{y}$ has dimension $mx1$, i.e., $(g(\boldsymbol{z}) - \boldsymbol{y}) \in \mathbb{R}^{mx1}$

- $\boldsymbol{x}_j$ has dimension $mx1$, i.e., $\boldsymbol{x} \in \mathbb{R}^{mx1}$

- To obtain conformant arguments for the product $(g(\boldsymbol{z}) - \boldsymbol{y})\boldsymbol{x}_j$, we can transpose $\boldsymbol{x}_j$, so that $\left(\boldsymbol{x}_j\right)^T \in \mathbb{R}^{1xm}$, and then rearrange terms:

$$slope_j = \frac{1}{m}\left(\boldsymbol{x}_j\right)^T(g(\boldsymbol{z}) - \boldsymbol{y})$$

Vector Product

- Expanding the vector product:

$$\frac{1}{m}\left(\boldsymbol{x}_j\right)^T\left(g(\boldsymbol{z})-\boldsymbol{y}\right)$$

$$\frac{1}{m}\left(\begin{bmatrix} x_j^{(1)} & x_j^{(2)} & \dots & x_j^{(m)} \end{bmatrix}\begin{bmatrix} g(z^{(1)})-y^{(1)} \\ g(z^{(2)})-y^{(2)} \\ \dots \\ g(z^{(m)})-y^{(m)} \end{bmatrix}\right)$$

Vectorized Implementation

- This is the *slope* for feature j,

$$slope_j = \frac{1}{m}(\boldsymbol{x}_j)^T(g(\boldsymbol{z}) - \boldsymbol{y})$$

- The list of slopes, formed by finding the slope for each feature, forms a gradient (i.e., vector of partial derivatives):

$$\boldsymbol{gradient} = \frac{1}{m}\begin{bmatrix} \boldsymbol{x}_0(g(\boldsymbol{z}) - \boldsymbol{y}) \\ \boldsymbol{x}_1(g(\boldsymbol{z}) - \boldsymbol{y}) \\ \ldots \\ \boldsymbol{x}_n(g(\boldsymbol{z}) - \boldsymbol{y}) \end{bmatrix}$$

Complete vector $\boldsymbol{x}$, i.e., all features of all training examples.

$$= \frac{1}{m}\boldsymbol{x}(g(\boldsymbol{z}) - \boldsymbol{y})$$

Dimensionality Analysis

- Consider the dimension of the vectors in $\boldsymbol{gradient}$:

$$\boldsymbol{gradient} = \frac{1}{m}\boldsymbol{x}(g(\boldsymbol{z}) - \boldsymbol{y})$$

- $(g(\boldsymbol{z}) - \boldsymbol{y})$ has dimension $mx1$, i.e.,$(g(\boldsymbol{z}) - \boldsymbol{y}) \in \mathbb{R}^{mx1}$
- $\boldsymbol{x}$ has dimension $mx(n+1)$, i.e., $\boldsymbol{x} \in \mathbb{R}^{mx(n+1)}$
- To obtain conformant arguments in the product $\boldsymbol{x}(g(\boldsymbol{z}) - \boldsymbol{y})$, we can transpose $\boldsymbol{x}$, so that $(\boldsymbol{x})^T \in \mathbb{R}^{(n+1)xm}$:

$$\boldsymbol{gradient} = \frac{1}{m}(\boldsymbol{x})^T(g(\boldsymbol{z}) - y)$$

Vector Product

- Expanding the vector product:

$$\frac{1}{m}(\boldsymbol{x})^T(g(\boldsymbol{z}) - y)$$

$$\frac{1}{m}\left(\begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ & & \dots & \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix}\begin{bmatrix} g(z^{(1)}) - y^{(1)} \\ g(z^{(2)}) - y^{(2)} \\ \dots \\ g(z^{(m)}) - y^{(m)} \end{bmatrix}\right) \in \mathbb{R}^{(n+1)x1}$$

Vectorized Implementation

- For vectorized implementation, the batch update rule is:

For each feature (j=0; j<=n; j++) {

$$\theta_j = \theta_j - \propto slope_j = \theta_j - \propto \frac{1}{m}\left(\boldsymbol{x}_j\right)^T (g(\boldsymbol{z}) - y)$$

}

- In vector format:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \propto \frac{1}{m}(\boldsymbol{x})^T (g(\boldsymbol{z}) - y)$$

Example 2

- Example 2 uses the data set of "Exercise 4: Logistic Regression and Newton's Method" from:

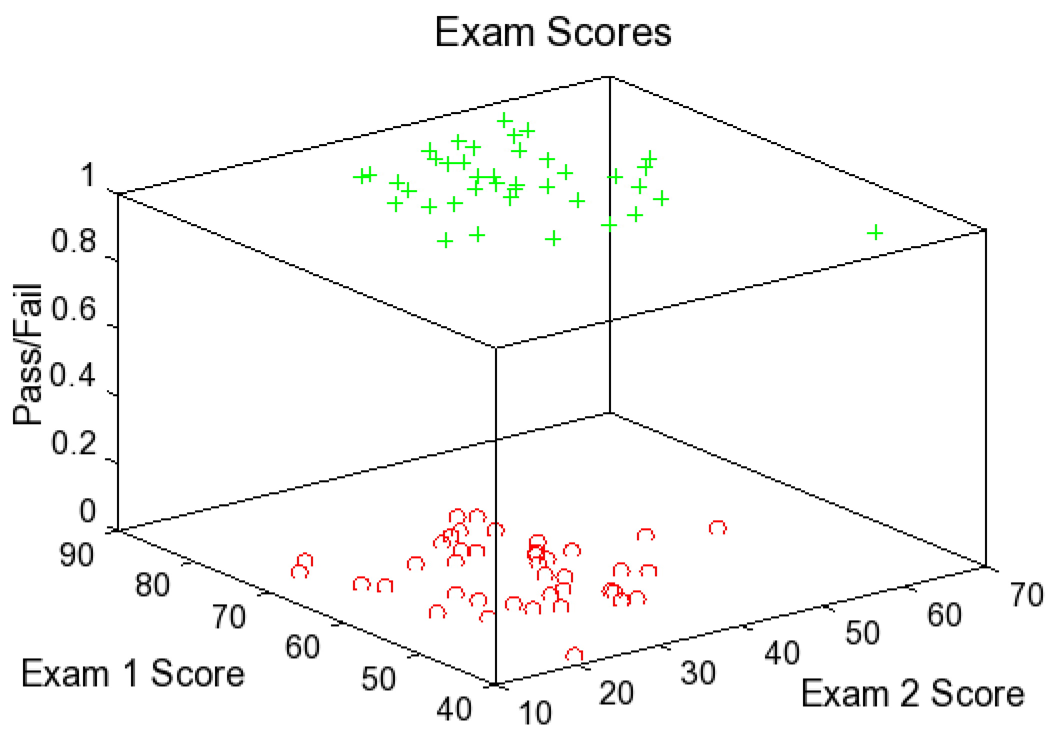
http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

- Example 2 compares the code and performance of the non-vectorized and the vectorized implementations of basic logistic regression.

Example 2

- This example uses a data set that represents scores on an exam in the first two columns (x_1 and x_2) the pass/fail mark in the 3rd column (y).
- 0 represents fail, and 1 represents pass.

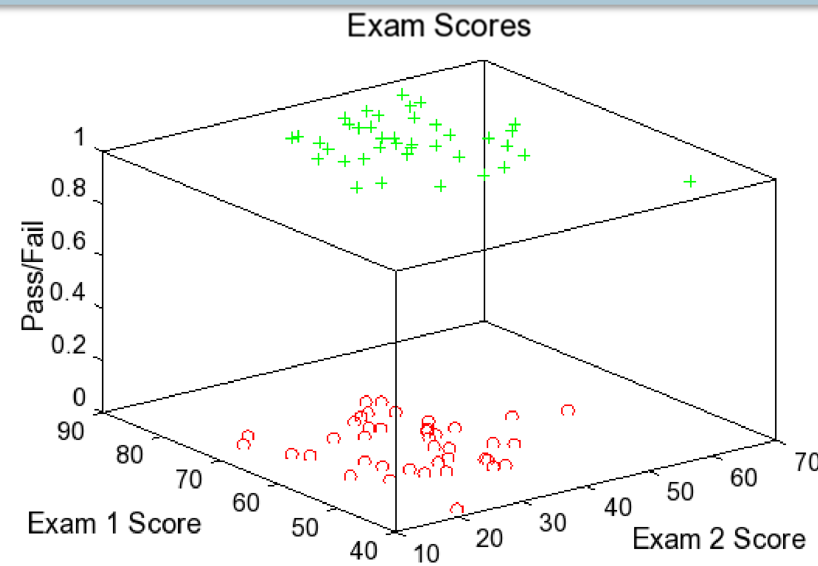
#	x_1	x_2	y
1	5.55E+01	6.95E+01	1
2	4.10E+01	8.15E+01	1
	…	…	1
78	1.85E+01	7.45E+01	0
79	1.60E+01	7.25E+01	0
80	3.35E+01	6.80E+01	0



Opening and Plotting Data Files

```
clear all; close all; clc
x = load('ex4x.dat'); y = load('ex4y.dat');
figure;
hold on
set(0, 'defaultaxesfontname', 'Arial');
set(0, 'defaultaxesfontsize', 16);

for i=1:length(y)
   if (y(i)==1)
      plot3(x(i,1),x(i,2),y(i),'+', 'color', 'g', 'markersize', 8);
   else
      plot3(x(i,1),x(i,2),y(i),'o', 'color', 'r', 'markersize', 8);
   endif
endfor
ylabel('Exam 1 Score', 'fontsize', 18, 'fontname', 'Arial');
xlabel('Exam 2 Score', 'fontsize', 18, 'fontname', 'Arial');
zlabel('Pass/Fail', 'fontsize', 18, 'fontname', 'Arial');
title('Exam Scores', 'fontsize', 20, 'fontname', 'Arial');
```



Non-vectorized Loop

Same as in Example 1

```
m = numTrainSam;
prevTheta=theta;
for t=1:maxIterations
      totError = 0;
      for j=1:numFeatures
         totSlope = 0;
         for i=1:m
            z=0;
            for jj=1:numFeatures
                z=z+prevTheta(jj)*x(i,jj);
            end
            h=1.0/(1.0+exp(-z));
            totSlope = (totSlope + (h-y(i))*x(i,j));
            totError = (totError + -y(i)*log(h)-(1-y(i))*log(1-h));
         end
         totError=totError/numTrainSam;
         theta(j)=theta(j)-learningRate*(totSlope/numTrainSam);
      end
      prevTheta=theta;
      errorPerIteration(t)=totError/numFeatures;
end
```

Vectorized Loop

$$\theta_j = \theta_j - \propto \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^i\right) x_j^{(i)}$$

$$h_\theta\left(x^{(i)}\right) = g\left(z^{(i)}\right) = \frac{1}{1 + e^{-\theta^T x^{(i)}}} = \frac{1}{1 + e^{-z^{(i)}}}$$

$$z^{(i)} = \theta^T x^{(i)} = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots \theta_n x_n^{(i)}$$

```
for t = 1:MAX_ITR
    % Update theta
    z = x * theta;
    h = g(z);
    grad = (1/m).*x' * (h-y);
    theta = theta - alpha .* grad;
    % Calculate J (for testing convergence)
    J(t) =(1/m)*sum(-y.*log(h) - (1-y).*log(1-h));
end
```

$z = x * theta$

x

1.0000	55.5000	69.5000
1.0000	41.0000	81.5000
1.0000	53.5000	86.0000
1.0000	46.0000	84.0000
1.0000	41.0000	73.5000
1.0000	51.5000	69.0000
1.0000	51.0000	62.5000
1.0000	42.0000	75.0000
1.0000	53.5000	83.0000
1.0000	57.5000	71.0000
1.0000	42.5000	72.5000
1.0000	41.0000	80.0000
1.0000	46.0000	82.0000
1.0000	46.0000	60.5000
1.0000	49.5000	76.0000
1.0000	41.0000	76.0000
1.0000	48.5000	72.5000
1.0000	51.5000	82.5000

…

theta

-0.051289
0.047569
-0.023269

=

z

0.9715479
0.0025683
0.4924646
0.1822382
0.1887239
0.7929078
0.9203748
0.2013884
0.5622730
1.0317811
0.2833464
0.0374725
0.2287771
0.7290702
0.5348843
0.1305502
0.5687586
0.4787703

…

$$z^{(i)} = \theta^T x^{(i)} = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots \theta_n x_n^{(i)}$$

- After 1000 iterations, theta is shown above. This is a suboptimal value of theta. A much better theta is achieved after 10,000,000 iterations.

$\boldsymbol{h = g(z)}$

$$g(\boldsymbol{z}) = \frac{1}{1 + e^{-\boldsymbol{z}}}$$

z	$h = g(\boldsymbol{z})$
0.9715479	0.72543
0.0025683	0.50064
0.4924646	0.62068
0.1822382	0.54543
0.1887239	0.54704
0.7929078	0.68845
0.9203748	0.71512
0.2013884	0.55018
0.5622730	0.63698
1.0317811	0.73726
0.2833464	0.57037
0.0374725	0.50937
0.2287771	0.55694
0.7290702	0.67460
0.5348843	0.63062
0.1305502	0.53259
0.5687586	0.63848
0.4787703	0.61745
…	…

$\boldsymbol{h} - \boldsymbol{y}$

$h = g(\mathbf{z})$		y		
0.72543		1		-0.27457
0.50064		1		-0.49936
0.62068		1		-0.37932
0.54543		1		-0.45457
0.54704		1		-0.45296
0.68845		1		-0.31155
0.71512		1		-0.28488
0.55018		1		-0.44982
0.63698		1		-0.36302
0.73726	$-$	1	$=$	-0.26274
0.57037		1		-0.42963
0.50937		1		-0.49063
0.55694		1		-0.44306
0.67460		1		-0.32540
0.63062		1		-0.36938
0.53259		1		-0.46741
0.63848		1		-0.36152
0.61745		1		-0.38255
…		…		…

$$\theta_j = \theta_j - \propto \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^i\right) x_j^{(i)}$$

$$h_\theta\left(x^{(i)}\right) - y^i$$

$x' * (h - y)$

x'

1.0000	1.0000	1.0000	1.0000	1.00
55.5000	41.0000	53.5000	46.0000	41.00
69.5000	81.5000	86.0000	84.0000	73.50

m Columns

*

-0.27457
-0.49936
-0.37932
-0.45457
-0.45296
-0.31155
-0.28488
-0.44982
-0.36302
-0.26274
-0.42963
-0.49063
-0.44306
-0.32540
-0.36938
-0.46741
-0.36152
-0.38255

...

m Rows

=

3.403316
-0.013470
-0.041932

$$\theta_j = \theta_j - \propto \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^i \right) x_j^{(i)}$$

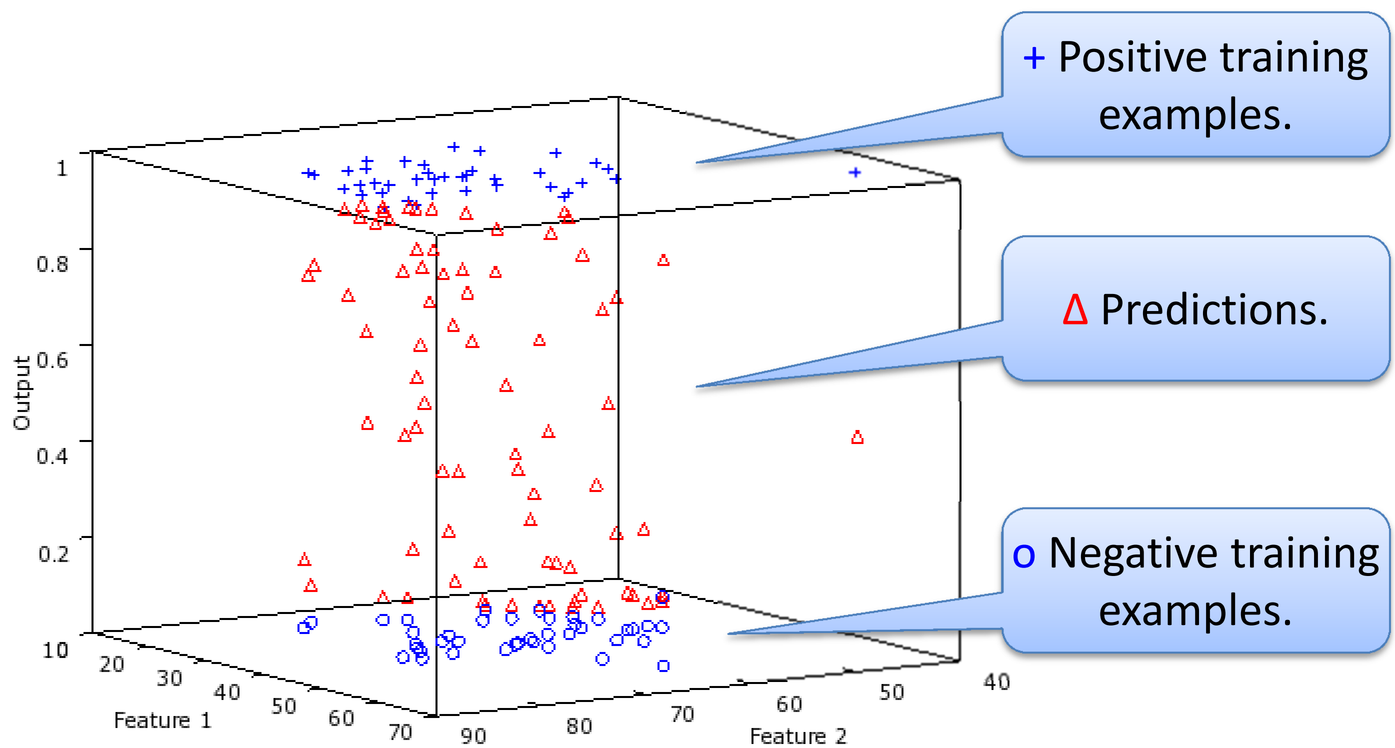
$$\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^i \right) x_j^{(i)}$$

Comparison: Non-vectorized vs Vectorized

	Non-vectorized	Vectorized
Iterations	100,000	
CPU Time (s)	4805.7	24.430
θ_1	-3.998494	-3.998494
θ_2	0.065799	0.065799
θ_3	0.024059	0.024059
Iterations	10,000,000	
CPU Time (s)	≈500,000	2532.5
θ_1		-16.37865
θ_2		0.14834
θ_3		0.15891

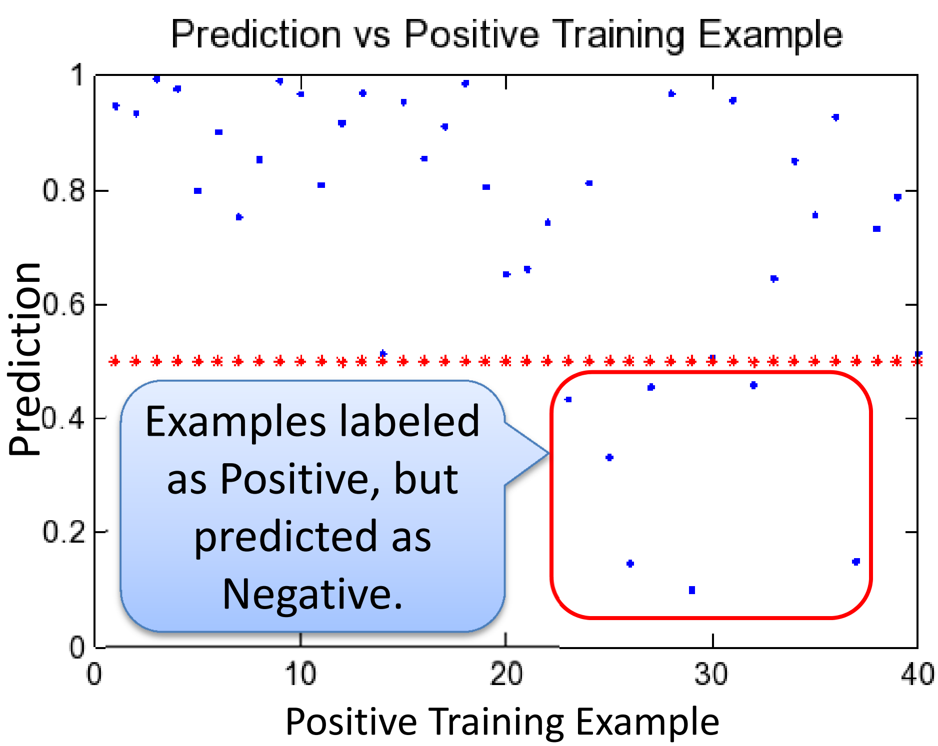
Visualizing Classification Result

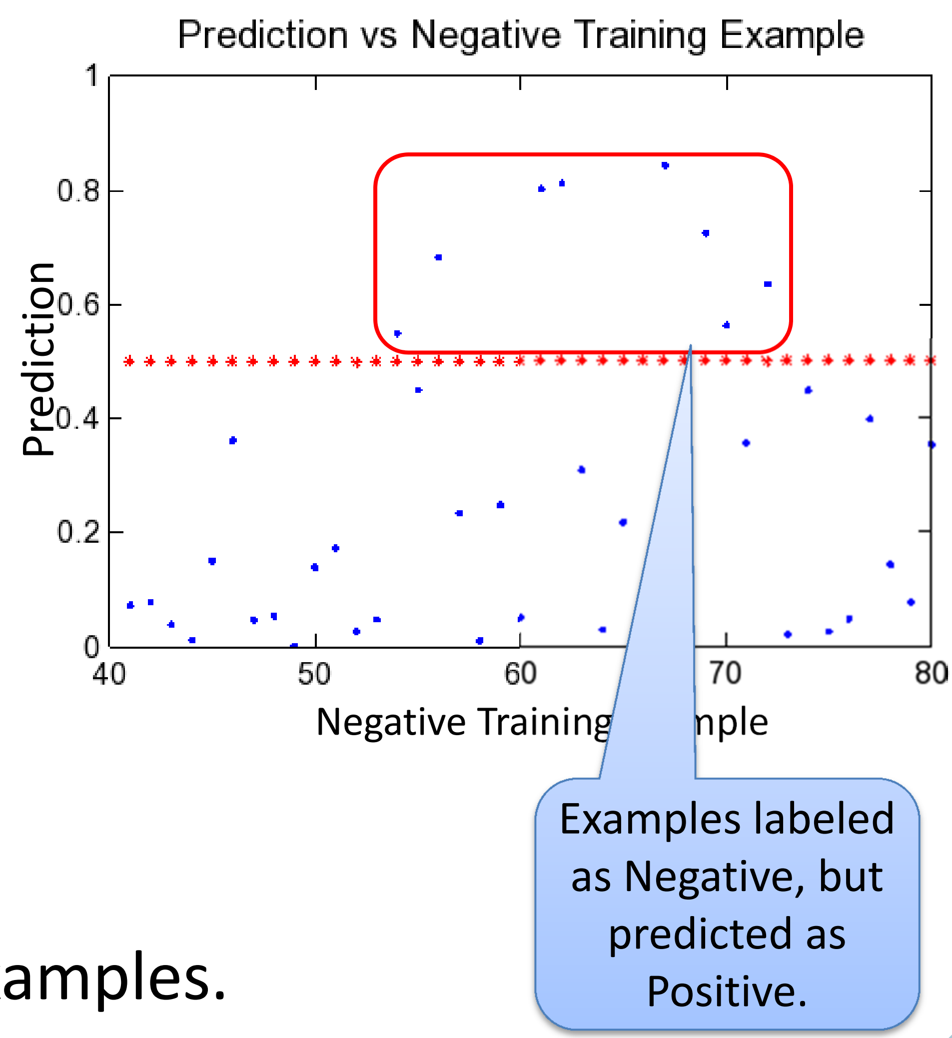
$$\theta = \begin{bmatrix} -16.380 \\ 0.1483 \\ 0.1589 \end{bmatrix}$$



- Predictions map to [0 1], and they represent the probability that the training example is positive.

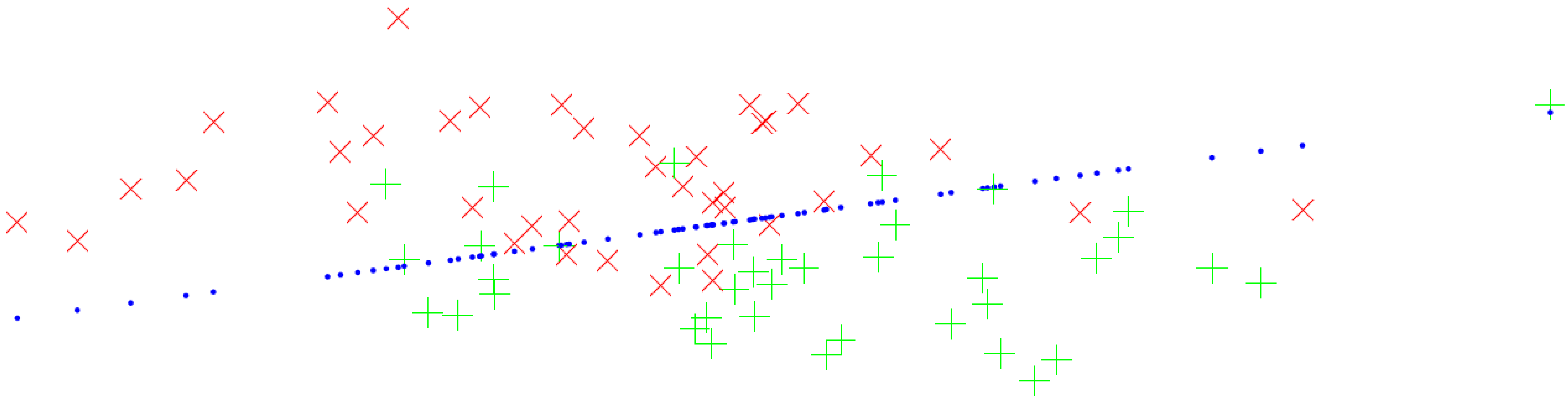
Training Performance



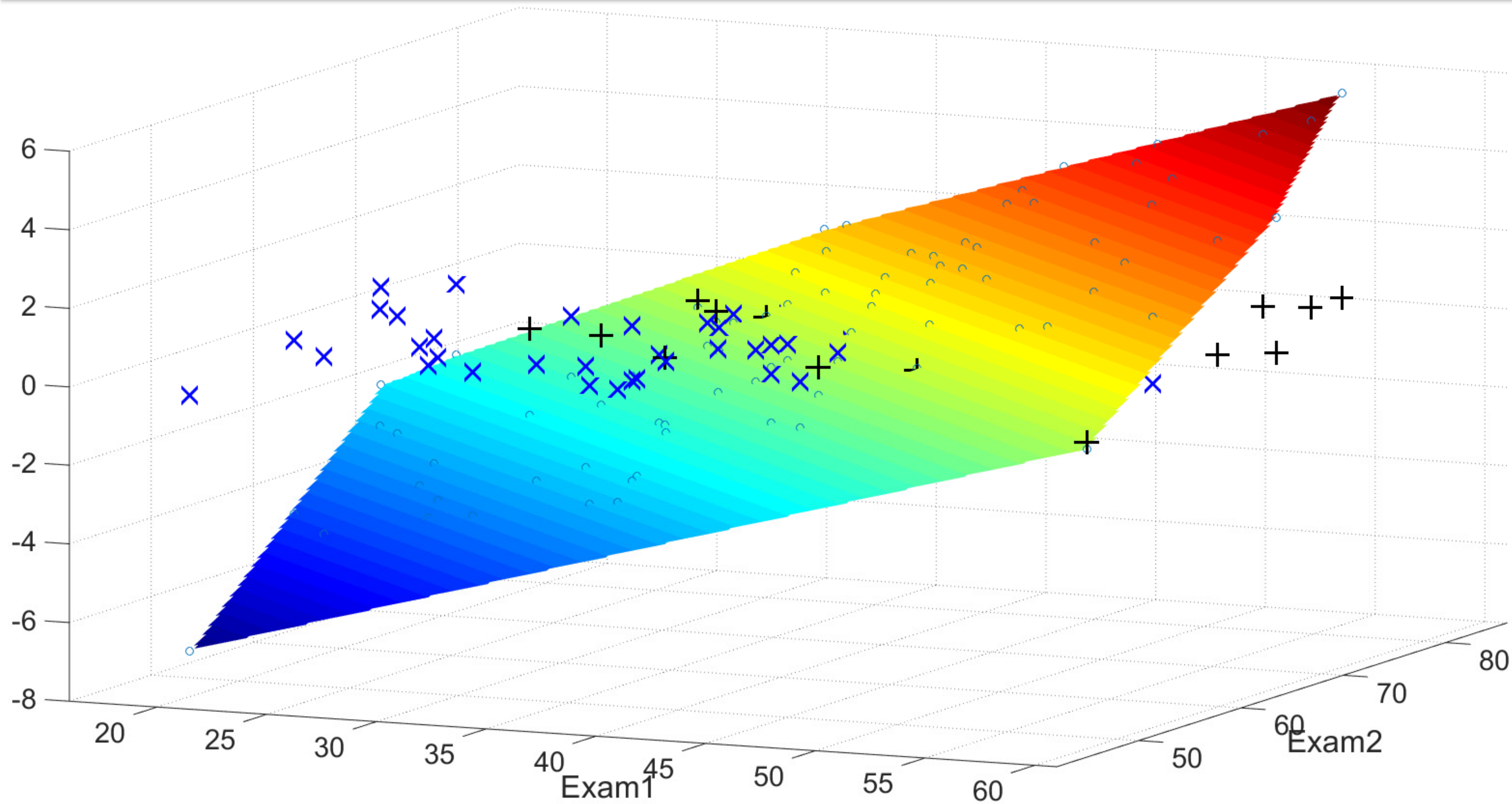


- Unable to learn 15 training examples.
- Training accuracy = 15/80 = 81.25%

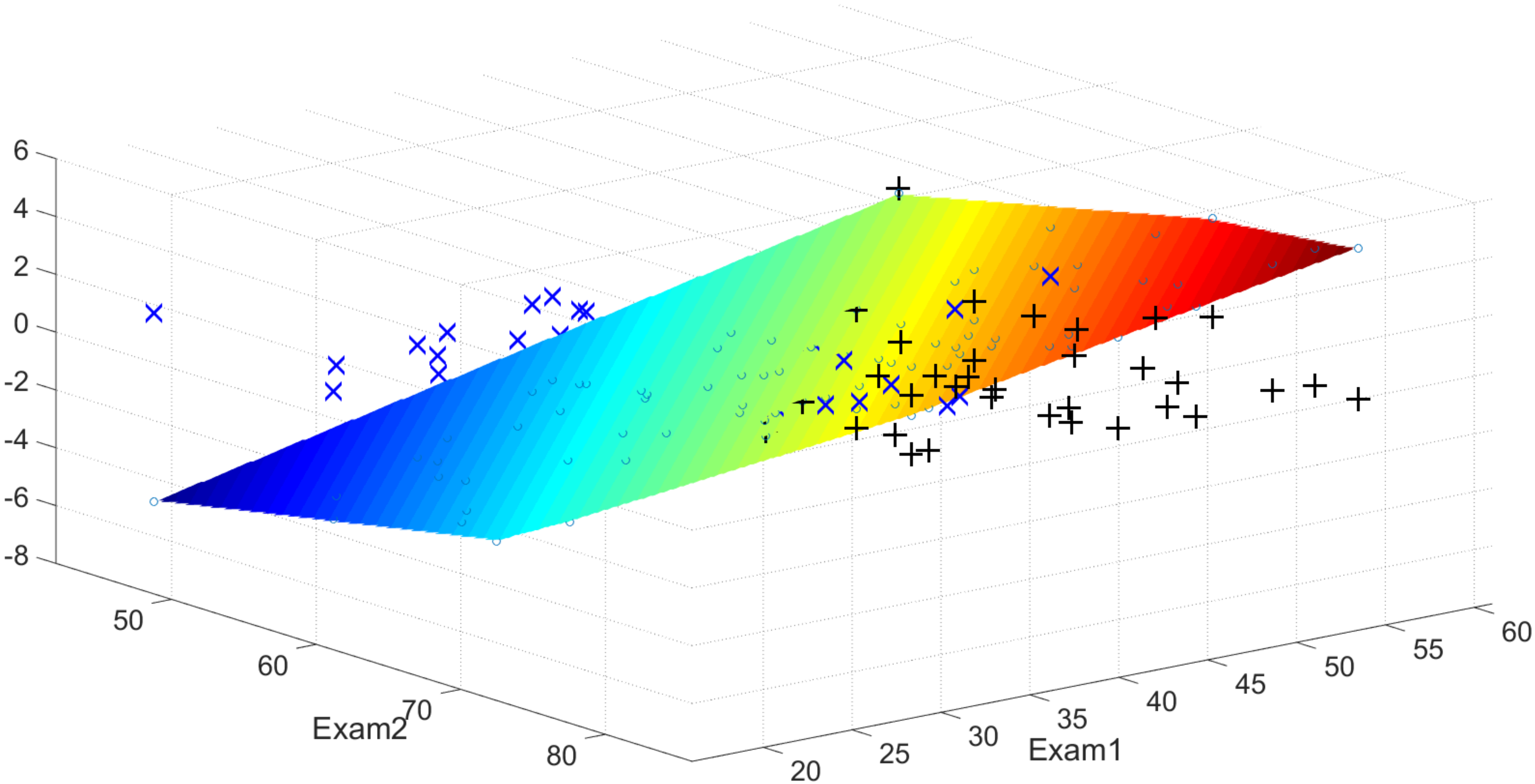
Training Performance



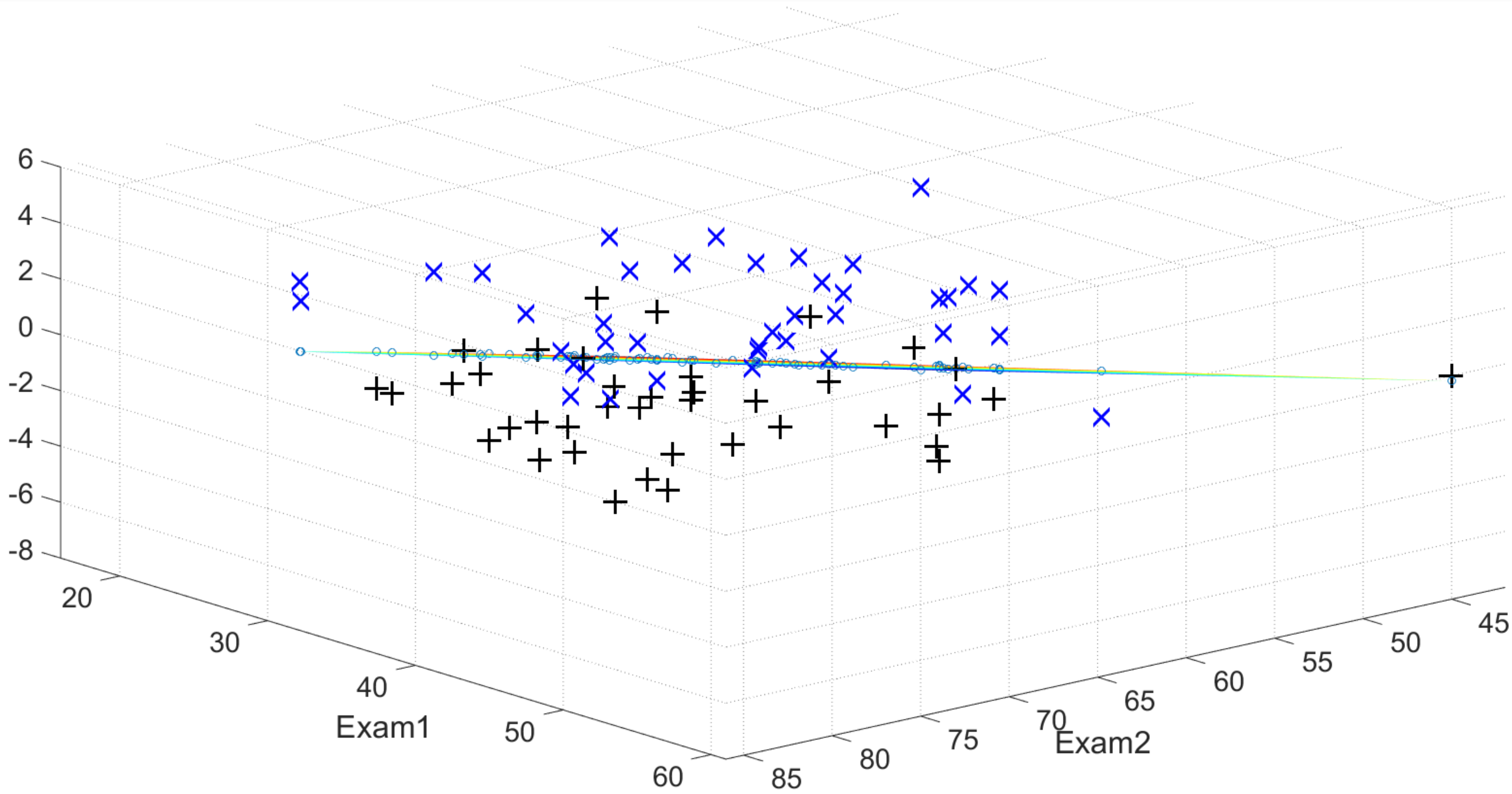
Training Performance



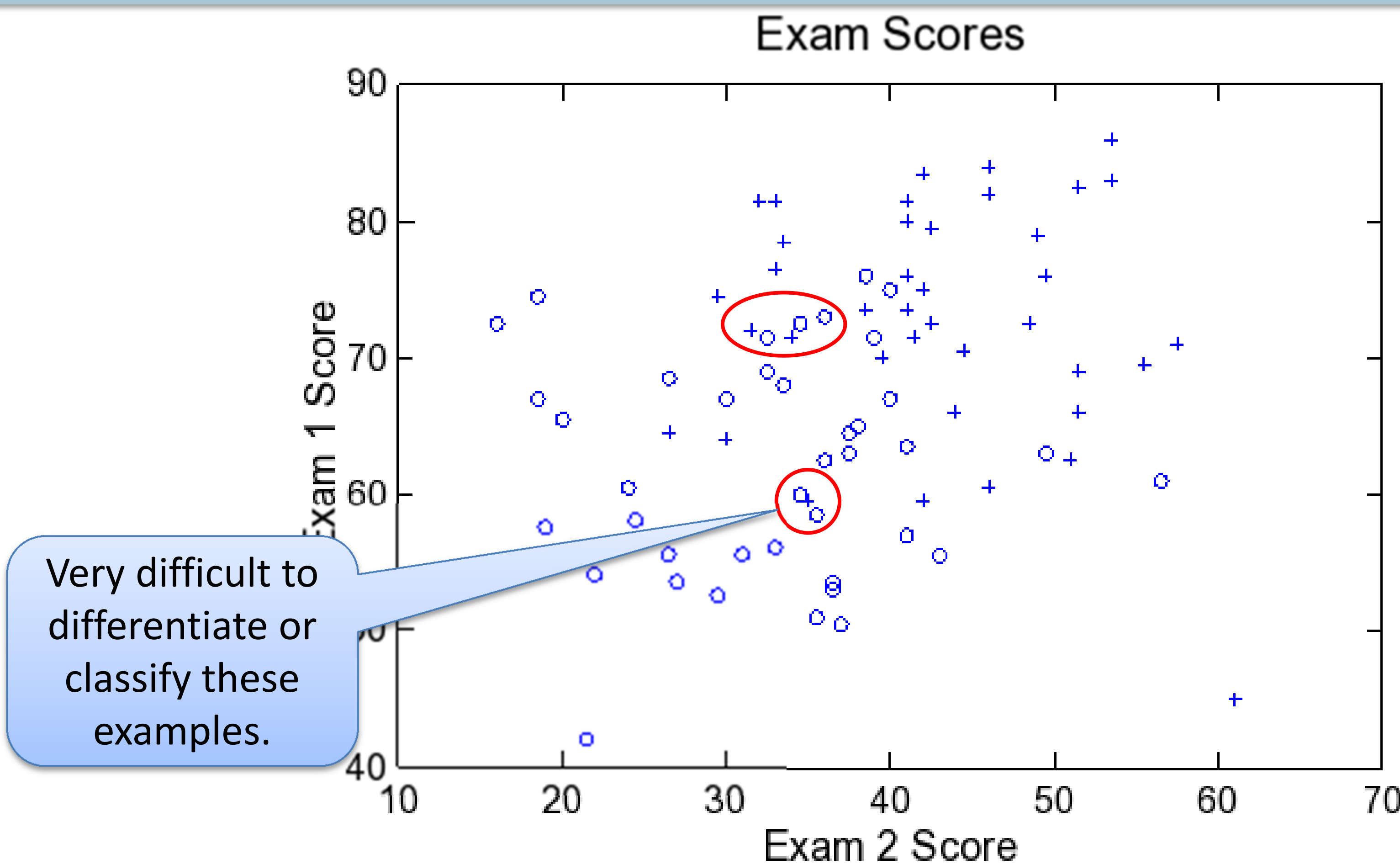
Training Performance



Training Performance



Training Performance



- Unable to differentiate some examples probably because they are very similar.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

[2] F. Rosenblatt, "The Perceptron--a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, New York, NY, 1957.

[3] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, Cambridge MA: The MIT Press, 1969.

[4] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," PhD thesis, Harvard University, Harvard, 1974.

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational properties," in Proceedings of the National Academy of Sciences of the USA, 1982.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

[2] F. Rosenblatt, "The Perceptron--a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, New York, NY, 1957.

[3] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, Cambridge MA: The MIT Press, 1969.

[4] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," PhD thesis, Harvard University, Harvard, 1974.

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational properties," in Proceedings of the National Academy of Sciences of the USA, 1982.