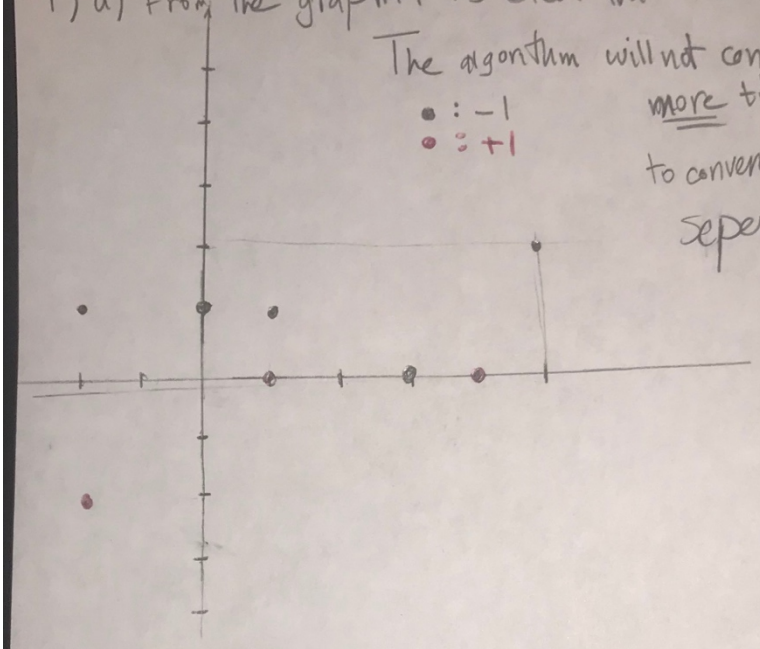
Milad Nourian
ECE M146, HW2

1) a) From the graph it is clear that data is no linearly seperable.



The algorithm will not converge if we run the algorithm more times. For the perceptron algorithm to converge, the data needs to be linearly seperable.

b) Algorithm for perceptron training:

$w \leftarrow 0, d \in \{0,1\}$
for iter in MaxIter:
  for (X, y) in data:
    $a \leftarrow \underline{w}^T \underline{x}$
    If $(a y_n < 0)$:
      $w^{t+1} \leftarrow w^t + y_n \underline{x}_n$
    else:
      # do nothing

(1) $w = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$ #initialize

1) $a = 16 \Rightarrow a y_1 > 0 \Rightarrow$ do nothing

(2) 2) $a = 4 \Rightarrow a y_2 < 0 \Rightarrow w = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + -1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ 2

3) $a = -1 \Rightarrow a y_3 > 0 \Rightarrow$ # do nothing

(4) 4) $a = -4 \Rightarrow a y_4 < 0 \Rightarrow w = \begin{bmatrix} 3 \\ -1 \end{bmatrix} + (1) \begin{bmatrix} -2 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$

5) $a = -5 \Rightarrow a y_5 > 0 \Rightarrow$ #nothing

6) $a = 1 \Rightarrow a y_6 > 0 \Rightarrow$ #nothing

7) $a = -1 \Rightarrow a y_7 > 0 \Rightarrow$ #nothing

(1) 8) $a = 3 \Rightarrow a y_8 < 0 \Rightarrow w = \begin{bmatrix} 1 \\ -3 \end{bmatrix} + (-1) \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ -3 \end{bmatrix}$

1)c) perceptron closed form: $\underline{w} = \begin{bmatrix} -2 \\ -3 \end{bmatrix} \rightarrow \hat{y} = \text{sign}(\underline{w}^T \underline{x})$

The algorithm for weighted perceptron:

Initialize $\underline{w} \leftarrow 0$, $\underline{u} \leftarrow 0$ ↳ includes bias

for iter = 1 in max iter:

  for $(x, y) \in D$:

    if $y_n(\underline{w}^T \underline{x}) \leq 0$: # update

      $\underline{w}^{(k+1)} \leftarrow \underline{w}^{(k)} + y_n \underline{x}_n$

      $\underline{u}^{(k+1)} \leftarrow \underline{u}^{(k)} + y c \underline{x}$

end if

We know: If $w^1, \ldots, w^{(k)} \Rightarrow w_1, \ldots, w_k$

$y_1, \ldots, y_n \quad c_1, \ldots, c_k$

$10 + 4 = 14$

$$w_{avg} = \sum c_k w_k$$

$$w_{avg} = 1\begin{bmatrix} 4 \\ 0 \end{bmatrix} + (2)\begin{bmatrix} 3 \\ -1 \end{bmatrix} + (4)\begin{bmatrix} 1 \\ -3 \end{bmatrix} + \begin{bmatrix} -2 \\ -3 \end{bmatrix}$$

using part a results $\underline{w}_{avg} = \left(\frac{1}{14}\right)\begin{bmatrix} 12 \\ -17 \end{bmatrix} \rightarrow \hat{y} = \text{sign}(\underline{w}_{avg}^T \underline{x})$

majority vote perceptron: take majority from $w_1, w_2, \ldots, w_k$

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^k \text{sign}(w^{(k)} \cdot \hat{x})\right)$$

$$\hat{y} = \text{sign}\left((1)\text{sign}\left(\begin{bmatrix} 4 \\ 0 \end{bmatrix}^T \cdot \underline{x}\right) + 2\,\text{sign}\left(\begin{bmatrix} 3 \\ -1 \end{bmatrix}^T \hat{x}\right) + 4\,\text{sign}\left(\begin{bmatrix} 1 \\ -3 \end{bmatrix}^T \hat{x}\right) + \text{sign}\left(\begin{bmatrix} -2 \\ -3 \end{bmatrix}^T \hat{x}\right)\right)$$

1) d) Find training error:

$W = \begin{bmatrix} -2 \\ -3 \end{bmatrix}$

$W_{avg} = \begin{bmatrix} 12 \\ -17 \end{bmatrix}$

$W_{majority} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$

	Perceptron	Average perceptron	Majority perceptron
1)	I	C	C
2)	C	C	C
3)	C	C	C
4)	C	C	C
5)	I	C	C
6)	I	C	C
7)	C	I	C
8)	C	I	I

Perceptron Accuracy = 0.625

Average perceptron Accuracy = 0.75

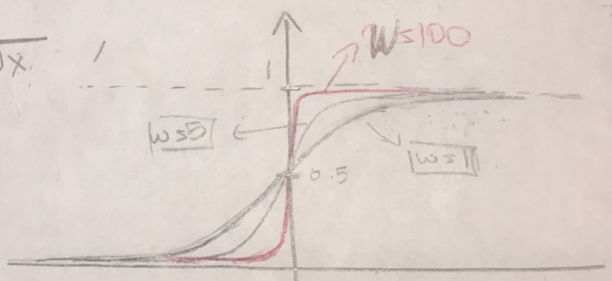
Majority " " = 0.875

Perceptron error = 0.375

Average perceptron error = 0.25

Majority perceptron error = 0.125

2) a) $\sigma(wx) = \frac{1}{1+e^{-wx}}$



For large values of w, the curve is steep. In this situation, a small change in the input results in a large change in the output which is an indication of overfit. To avoid overfitting, we would want the curves to be smooth as possible, meaning a small change in the input would result in a small change in the output.

2b) $J(w) = -\sum_{n=0}^{N} y_n \log h_w(x_n) + (1-y_n)\log(1-h_w(x_n)) + \frac{1}{2}\sum_i w_i^2$

$$= -\sum_{n=0}^{N} (y_n)\log(\underbrace{w^T x}_{-\log(1+e^{-w^T x})}) + (1-y_n)\log(1-w^T x_n) + \frac{1}{2}\sum_i w_i^2$$

$$\frac{\partial J}{\partial w_j} = -\sum_{n=0}^{N} y_n \frac{\partial}{\partial w_j}\log(w^T x) + (1-y_n)\frac{\partial}{\partial w_j}\log(1-w^T x_n) + \frac{1}{2}\sum_i \frac{\partial}{\partial w_i} w_i^2$$

$$= -\sum_{n=0}^{N} (y_n)\left[\frac{1}{1+e^{-w^T x}}\right][x_j] + (1-y_n)\left[\frac{1}{1+e^{-(1-w^T x_n)}}\right][-x_j] + w_i$$

$$= (-x_j)\sum_{n=0}^{N}\left[\frac{-y_n}{1+e^{-w^T x}} + \frac{(1-y_n)}{1+e^{-(1-w^T x_n)}}\right] + w_i$$

$$\boxed{\nabla G(w) = \sum_n \left[w^T x - y_n\right] x_n + w} \quad \checkmark$$

This is the gradient descent step for linear Regression now

$$w^{t+1} \leftarrow w^t - \eta\left[\sum_n \left(w^T x - y_n\right) x_n + w\right] \Rightarrow \text{So now we add } w$$

2)c) We know
We know every (x_i, y_i) pair is IId, so

$$\text{Log}L(w) = \log\left[\prod_{i=1}^{n} P(y_i|x_i,\theta)\, f(w_0, \dots, w_n)\right]$$

$$= \sum_{i=1}^{n} \log P(y_i|x_i,\theta) + \log f(w_0, \dots, w_n)$$

$$= \sum_{i=1}^{n} \log\left(\left[h_\theta(x_n)\right]^{y_n}\left[1-h_\theta(x_n)\right]^{1-y_n}\right) + \log\frac{1}{\sqrt{2\pi}} e^{-\frac{(\sum w_i^2)}{2}}$$

$$= \sum_{i=1}^{n} y_n \log h_\theta(x_n) + (1-y_n)\left[1-h_\theta(x_n)\right] + \log 2\pi - \sum_i \frac{w_i^2}{2}$$

$$\text{Log}L(w) = \sum_{i=1}^{n} y_n \log h_\theta(x_n) + \left[1-y_n\right]\left[1-h_\theta(x_n)\right] - \log 2\pi - \sum_i \frac{w_i^2}{2}$$

Found this in class:

Comparing $\text{Log}L(w)$ with $J(w)$ given in the problem, we see that

$$\text{Log}L(w) = -J(w)$$

And in part a, we found $\frac{\partial J(w)}{\partial w}$. So, we can conclude that

Minimizing the cost function is same as maximizing likelihood $L(w)$.

3)a)

$$J(w_0, w_1) = \sum_n \alpha_n \left[w_0 + w_1 x_{n,1} - y_n \right]^2$$

$$\frac{\partial J}{\partial w_0} = \sum_n \left[(\alpha_n)(2)\left(w_0 + w_1 x_{n,1} - y_n\right)[1] \right] = 2\sum_n [\alpha_n]\left[w_0 + w_1 x_{n,1} - y_n \right]$$

$$\frac{\partial J}{\partial w_1} = 2\sum_n [\alpha_n]\left[w_0 + w_1 x_{n,1} - y_n \right][x_{n,1}]$$

so now find gradient $\nabla J = \begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \end{bmatrix} = 2\sum_n [a_n]\left[\underline{w}^T \underline{x} - y_n \right] \underline{x}_n$

*So the only change is that now we have a coefficient α_n (as weight).

b) We can first calculate Hessian Matrix:

$$H = \begin{bmatrix} \frac{\partial J}{\partial w_0^2} & \frac{\partial J}{\partial w_0 \partial w_1} \\ \frac{\partial J}{\partial w_1 \partial w_0} & \frac{\partial J}{\partial w_1^2} \end{bmatrix} = \begin{bmatrix} 2\sum_n a_n & 2\sum_n \alpha_n x_{n,1} \\ 2\sum_n [a_n][x_{n,1}] & 2\sum_n [a_n][x_{n,1}]^2 \end{bmatrix}_{2\times 2}$$

Now, we know If $z^T H z > 0 \Rightarrow$ then convex $\quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_{2\times 1}$

$$2\begin{bmatrix} \sum_n a_n z_1 + \sum a_n x_{n,1} z_2 & \sum_n a_n x_{n,1} z_1 + \sum_n a_n (x_{n,1})^2 z_2 \end{bmatrix}_{1\times 2} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$= \left\{ \sum_n a_n z_1^2 + \sum a_n x_{n,1} z_2 (z_1) + \sum_n a_n x_{n,1} z_1 z_2 + \sum a_n (x_{n,1})^2 z_2^2 \right\}$$

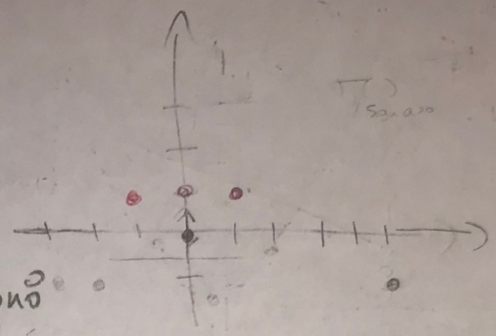
$$= \left[\sum a_n\right]\left[z_1^2 + 2x_n z_1 z_2 + x_n^2 z_2^2 \right] > 0 \Rightarrow \text{so } J(w) \text{ is convex and}$$

$\boxed{(z_1 + x_n z_2)^2 \geq 0}$ ✓

therefore, there exists a global minimum (global min = local min)

4)a)

$x_1 = -1$	$\rightarrow y = +1$
$x_2 = 0$	$\rightarrow y = +1$
$x_3 = 1$	$\rightarrow y = +1$
$x_4 = -3$	$\rightarrow y = -1$
$x_5 = -2$	$\rightarrow y = -1$
$x_6 = 3$	$\rightarrow y = -1$



One optimal solution:

$$\underline{w}_{opt} = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix} \quad / \text{Accuracy} = \frac{5}{6}$$

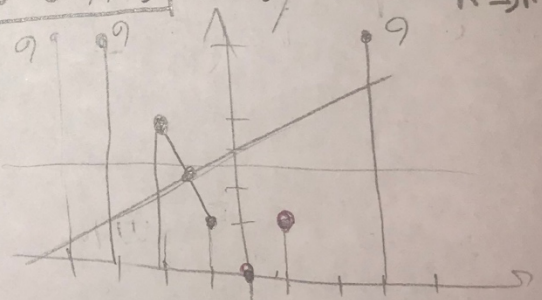
b) $K(x,z) = xz(1+xz)$

$= xz + (xz)^2$

$\Rightarrow$ Therefore feature map $\boxed{\phi(x) = (x, x^2)}$ transforms from $R^1 \rightarrow R^2$

Now calculating $\phi(x)$ for all points

$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\begin{bmatrix} -3 \\ 9 \end{bmatrix}$ $\begin{bmatrix} -2 \\ 4 \end{bmatrix}$ $\begin{bmatrix} 3 \\ 9 \end{bmatrix}$



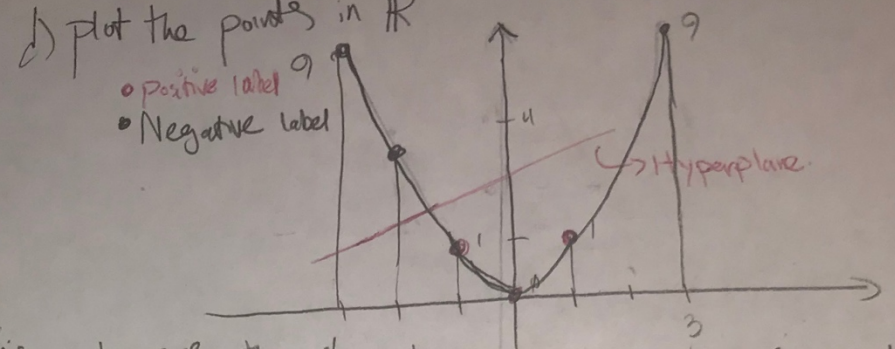
Answer: We can see that the line must pass between $V_1 = [-2, 4]$ and $V_2 = [-1, 1]$ to seperate the data.

The seperating line can also go through middle of V_1, V_2.

So we can get $-Y_1 + 3Y_2 - 9 = 0$

$$\underline{w} = \begin{bmatrix} -1 \\ 3 \\ -9 \end{bmatrix} \Rightarrow \text{is one optimal solution.}$$

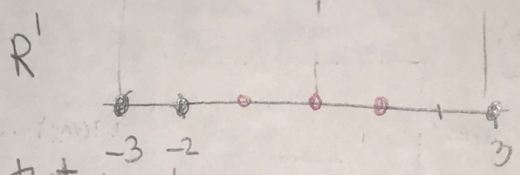
d) plot the points in $\mathbb{R}^2$



This graph agrees with our intuition, as $\phi(x) = (x, x^2)$ was our feature map and we expected a prabola (polynomial).

Optimal $\underline{w}$ ($\underline{w}_{opt}$) has accuracy 1.0 as it correcly classifies all the points. $\Rightarrow$ Accuracy = 1.0

e)



We can see that the data in not linearly seperable in $\mathbb{R}^1$.

ECE M146, HW 2

5)a)

- We can see from the data the scatter plot, it seems to follow a cubic behavior. From this behavior we can expect that using linear regression to model this data would most likely **underfit**, as we need to increase the order of polynomial (M) in our model to better fit the given data that we see from the scatter plot.

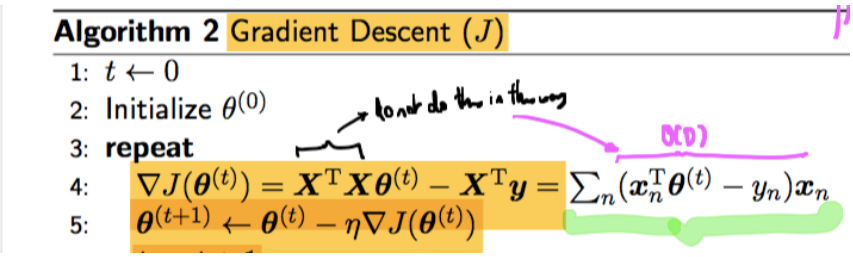
5)d)

For this table I iterated through all the input points and updated the theta vector using the following formula (used 2 in the gradient descent formula):

$$w_j \leftarrow w_j - 2\alpha \sum_{n=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}_n) - y_n\right) x_{n,j} \quad \text{(simultaneously update } w_j \text{ for all } j\text{)}$$

: Coefficients	: Number of iterations	: Objective Function
[2.27040138 -2.46075209]	: 10000	: 4.08633386426
[2.44566862 -2.81750141]	: 10000	: 3.91261070545
[2.43797706 -2.82790485]	: 1233	: 3.91658245895
[2.39884098 -2.86600362]	: 332	: 4.0160981729

For this table I used the matrix formula given to us in the class as shown below (did not use the 2 coefficient):



For each of the step values 10^-4, 10^-3, 10^-2, 0.0407

: Coefficients	: Number of iterations	: Objective Function
[1.91573585 -1.74358989]	: 10000	: 5.49356558874
[2.4463815 -2.81630184]	: 10000	: 3.91257640947
[2.44640698 -2.81635335]	: 1480	: 3.91257640579
[2.44640705 -2.81635349]	: 377	: 3.91257640579

- The coefficients for the case with small step size (10^-4) are smaller than cases with larger step size. Basically, we see a trend that increasing the step size will make the algorithm for linear regression to converge faster. Also, increasing the step size will gives larger weight values (coefficients), so smaller step sizes have smaller coefficients and larger step sizes have larger coefficient values. For step size of 10^-4 and 10^-3, we never converge suggesting that the movement has been too slow, but for the step size of 10^-2, the algorithm for the gradient descent converges after 1480 iterations and for step size of 0.0407, we converge after 377 iterations. In addition, it can be seen that the

cost function (J, or objective function) for small step size is large, and it gets smaller as the step size get increases.

5)e)

- The coefficients for theta using the **close form solution** is found to be:

: Coefficients	:	Objective Function
: [2.44640709 -2.81635359]	:	3.91257640579

Comparing with the solution we obtained from the Gradient descent algorithm, we can see that the coefficients and the cost function from Gradient descent (for step size 0.0407 and 10^-2) are identical to the values for coefficients and cost found in the closed form. Therefore, we can see that when the gradient descent converges, it reaches the same answer as the closed form solution for the optimal theta, and our expectation holds true.

The closed-form answer found takes longer than the gradient descent approach. The reason for this is that finding the inverse of matrix $(X^TX)^{-1}$ takes $O(D^3)$, which computationally heavy due to its high order complexity whereas updating theta in the gradient descent takes O(ND) on each iteration and therefore, has less complexity.

5)f)

by changing the value of the step size on each iteration, the following result was obtained:

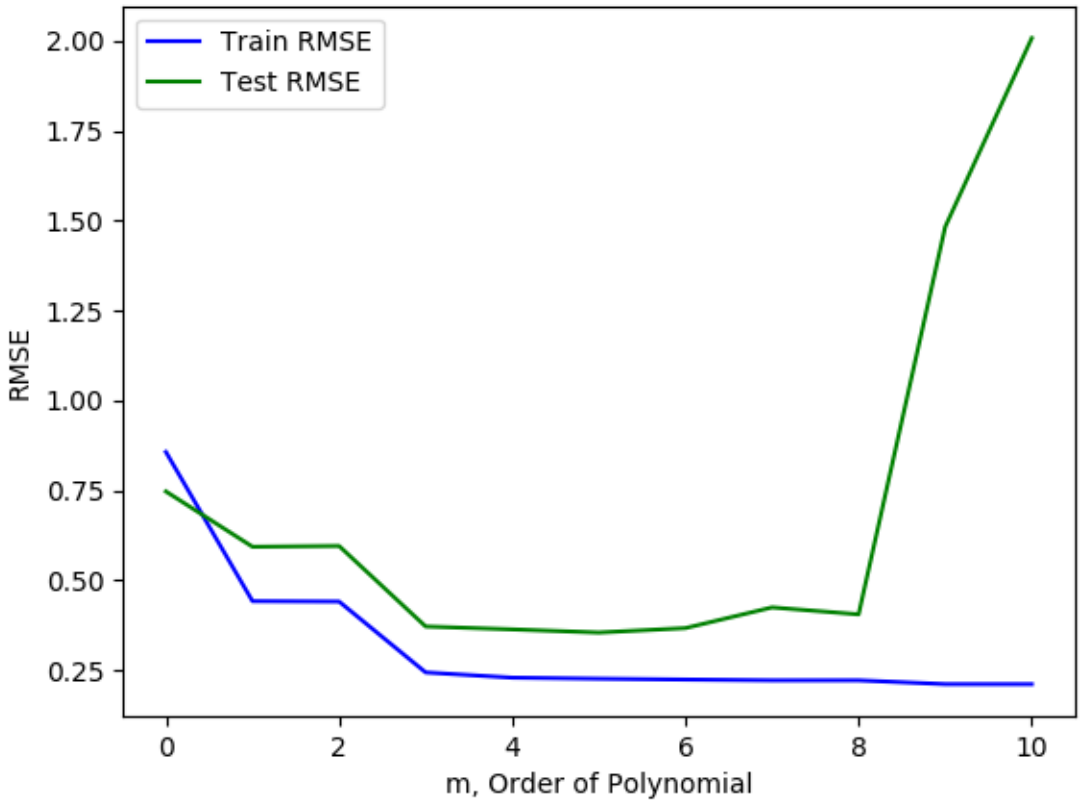
: Coefficients	: Number of iterations	: Objective Function
[2.44635846 -2.81652261]	: 10000	: 3.91257677039

It can be seen from the above result that by changing the value of step size, the algorithm does converge but it takes all 10000 iterations. This can be explained because the algorithm uses step = 1/ (1+k), so it starts with big step sizes and toward the end the step size becomes very small, taking very long for the algorithm to converge.

5)h)

- One reason why MRS is used instead of regular J(w) (cost function) is that MRS provides a normalized value, where the number of samples N is already being taken into the account. This can eliminate the performance measurement dependency on the input data. Using MRS instead of J(w) is a better representative of performance measurement.

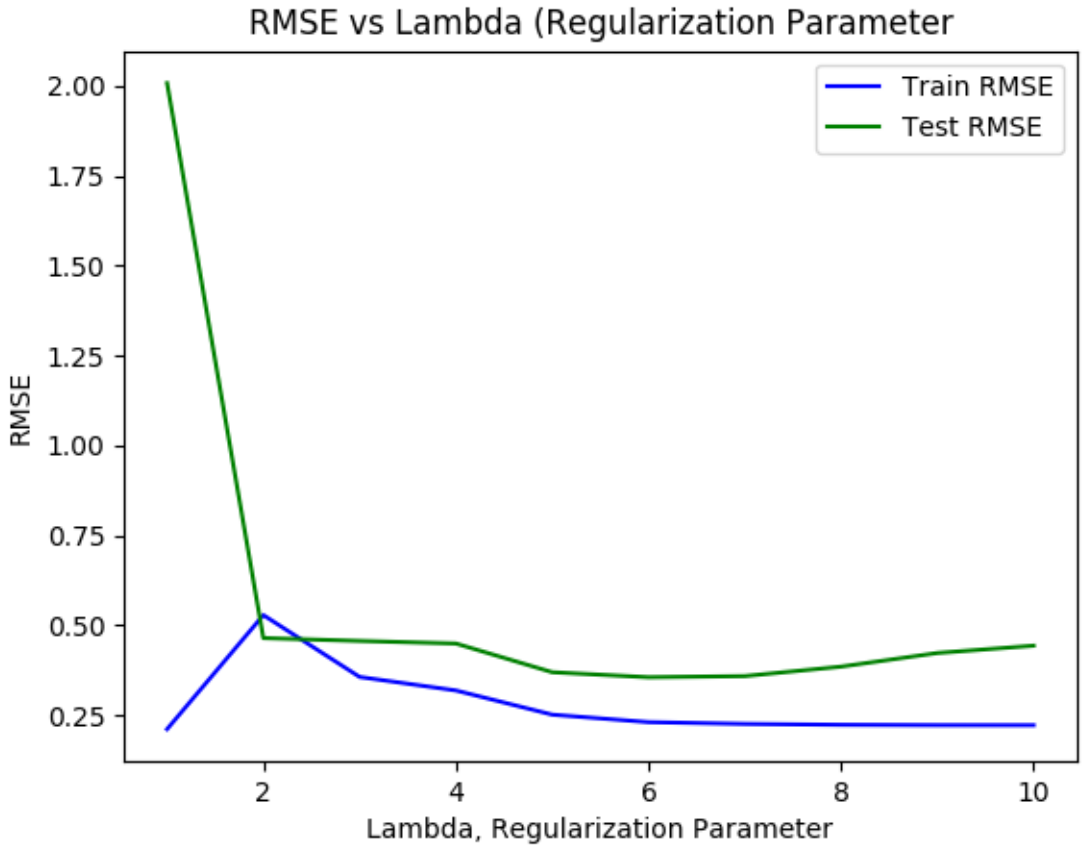
5)i)



- From the diagram, we can see that m = 3 best fits the data due to the following reasons:
    - For m = 3, the test error is at minimum
    - When visualizing data in 5)a) we also anticipated the polynomial fit to be a cubic
    - m = 3 is relatively a low degree polynomial, and to avoid overfitting, we tend to use the "simplest models" that fit the data pretty well, so in this case m = 3

- There can be seen a significant amount of overfitting for large values of m. From the MSRE graph we see that by increasing m for m>=3, the training error reduces meaning that our model fits the training data very well, however, as m increases further, the test error increases very rapidly, showing a case of overfitting.

5)k)



- From the graph, we can see that the best value for λ is 5 because this value minimizes the RMSE.