Danning Yu, 305087992
CS M146 Discussion 1A
PS2
2/2/2020

1. (a) Using T = 1 and F = –1, we create the following "truth" table for Y = OR(X_1, X_2)

X_1	X_2	Y
–1	–1	–1
–1	1	1
1	–1	1
1	1	1

Plotting this a graph with X_1 as the horizontal axis and X_2 as the vertical axis gives points with value 1 in the first, second, and fourth quadrants, while the point in the third quadrant has value –1. We can draw a line that separates the –1 data point from the +1 data point. Thus, it is linearly separable. If we let $\boldsymbol{\theta} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T$, then values that work are $\boldsymbol{\theta} = \begin{bmatrix} 0.75 & 1 & 1 \end{bmatrix}^T$ and $\boldsymbol{\theta} = \begin{bmatrix} 0.5 & 1 & 1 \end{bmatrix}^T$.

(b) We create the following "truth" table for Y = XOR(X_1, X_2):

X_1	X_2	Y
–1	–1	–1
–1	1	1
1	–1	1
1	1	–1

If we plot this as we did with 1(a), we see that there is no way to draw a line that separates the +1 and –1 data points, so thus, no perceptron exists. Alternatively, we get the following equations:

$$\theta_0 - \theta_1 - \theta_2 < 0$$
$$\theta_0 - \theta_1 + \theta_2 > 0$$
$$\theta_0 + \theta_1 - \theta_2 > 0$$
$$\theta_0 + \theta_1 + \theta_2 < 0$$

Combining the first and fourth equations gives $2\theta_0 < 0$, while combining the second and third equations gives $2\theta_0 > 0$, so the system of equations has no solution.

2. (a) Given $J(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\left[y_n \log h_{\boldsymbol{\theta}}(\mathbf{x}_n) + (1-y_n)\log\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right)\right]$, where $h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(a(\mathbf{x})\right)$ and $a(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x} + b$.

First, get the derivative of $a(\mathbf{x})$ with respect to θ_j:

$$\frac{da}{d\theta_j} = \frac{d}{d\theta_j}\left(\boldsymbol{\theta}^T\mathbf{x} + b\right) = x_j$$

Then, find the derivative of $\sigma(a)$ (note: leaving out the a' term that arises from the chain rule):

$$\frac{d\sigma}{da} = \frac{d}{da}\left(\frac{1}{1+e^{-a}}\right) = \frac{0-(1)\left(0+e^{-a}\right)(-1)}{\left(1+e^{-a}\right)^2} = \frac{e^{-a}}{\left(1+e^{-a}\right)^2} = \left(\sigma(a)\right)\left(1-\sigma(a)\right)$$

Now use these two above results to find the desired answer:

$$\frac{\partial J}{\partial\theta_j} = \frac{\partial}{\partial\theta_j}\left(-\sum_{n=1}^{N}\left[y_n \log h_{\boldsymbol{\theta}}(\mathbf{x}_n) + (1-y_n)\log\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right)\right]\right)$$

$$= -\sum_{n=1}^{N}\left[y_n \frac{\partial}{\partial\theta_j}\left(\log h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right) + (1-y_n)\frac{\partial}{\partial\theta_j}\left(\log\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right)\right)\right]$$

$$= -\sum_{n=1}^{N}\left[y_n \frac{1}{\sigma(a)}\frac{\partial}{\partial\theta_j}\left(\sigma(a)\right) + (1-y_n)\frac{1}{1-\sigma(a)}\frac{\partial}{\partial\theta_j}\left(1-\sigma(a)\right)\right]$$

$$= -\sum_{n=1}^{N}\left[y_n \frac{1}{\sigma(a)}\sigma(a)\left(1-\sigma(a)\right)\frac{\partial}{\partial\theta_j}\left(a(\mathbf{x}_n)\right) + (1-y_n)\frac{1}{1-\sigma(a)}(-1)\sigma(a)\left(1-\sigma(a)\right)\frac{\partial}{\partial\theta_j}\left(a(\mathbf{x}_n)\right)\right]$$

$$= -\sum_{n=1}^{N}\left[y_n\left(1-\sigma(a)\right)x_{n,j} - (1-y_n)\sigma(a)x_{n,j}\right]$$

$$= -\sum_{n=1}^{N}\left(y_n - \sigma(a)\right)x_{n,j}$$

$$= \sum_{n=1}^{N}\left(h_{\boldsymbol{\theta}}(\mathbf{x}_n) - y_n\right)x_{n,j}$$

(b) To get second partial derivatives, take partial derivative of 2(a) with respect to θ_k:

$$\frac{\partial^2 J}{\partial\theta_k\partial\theta_j} = \frac{\partial}{\partial\theta_k}\left(\sum_{n=1}^{N}\left(h_{\boldsymbol{\theta}}(\mathbf{x}_n) - y_n\right)x_{n,j}\right) = \sum_{n=1}^{N}\frac{\partial}{\partial\theta_k}\left(h_{\boldsymbol{\theta}}(\mathbf{x}_n) - y_n\right)x_{n,j}$$

$$= \sum_{n=1}^{N}h_{\boldsymbol{\theta}}(\mathbf{x}_n)\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right)x_{n,j}x_{n,k}$$

Extending this to all *j*, *k* gives us all the entries in the Hessian matrix and the desired result, which can thus be written as:

$$\mathbf{H} = \sum_{n=1}^{N}h_{\boldsymbol{\theta}}(\mathbf{x}_n)\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_n)\right)\mathbf{x}_n\mathbf{x}_n^T$$

(c) A Hessian is positive semi-definite (PSD) if $\mathbf{z}^T\mathbf{Hz} \geq 0$ for all real vectors **z**. Writing it in summation form gives the following:

$$\mathbf{z}^T\mathbf{Hz} = \sum_{n=1}^{N} h_{\boldsymbol{\theta}}\left(\mathbf{x}_n\right)\left(1-h_{\boldsymbol{\theta}}\left(\mathbf{x}_n\right)\right)\mathbf{z}^T\mathbf{x}_n\mathbf{x}_n^T\mathbf{z}$$

Now note that $\mathbf{x}_n^T\mathbf{z} = \left(\mathbf{z}^T\mathbf{x}_n\right)^T$ and that the expression $\mathbf{x}_n^T\mathbf{z}$ evaluates to a scalar. If we let this scalar be *p*, we have $p^T = p$, so thus, $\mathbf{z}_n^T\mathbf{x}_n\mathbf{x}_n^T\mathbf{z}_n = p^2 \geq 0$. Then, $h_{\boldsymbol{\theta}}\left(\mathbf{x}_n\right)$ is restricted to the range [0, 1] due to the property of the sigmoid function, and thus so is $\left(1-h_{\boldsymbol{\theta}}\left(\mathbf{x}_n\right)\right)$. Thus, the product of all the terms in the summation will be positive, meaning that $\mathbf{z}^T\mathbf{Hz} \geq 0$ and that the Hessian has been shown to be PSD, which means that *J* is a convex function, as desired.

3. (a) For a Bernoulli distribution, assuming independent and identically distributed (IID),

$$L(\theta) = P(x_1;\theta)P(x_2;\theta)\ldots = \prod_{i=1}^{n} P(x_i;\theta) = \prod_{i=1}^{n} P\left(\theta^{x_i}(1-\theta)^{1-x_i}\right)$$

Because the variables are IID, the likelihood function does not depend on the order that the random variables are observed.

(b) Taking the log of both sides to get the log likelihood:

$$l(\theta) = \log\left(\prod_{i=1}^{n} P(x_i;\theta)\right) = \sum_{i=1}^{N} \log P(x_i;\theta) = \sum_{i=1}^{N} \log P\left(\theta^{x_i}(1-\theta)^{1-x_i}\right)$$

$$= \sum_{i=1}^{N}\left[x_i \log\theta + (1-x_i)\log(1-\theta)\right]$$

First and second derivatives:

$$\frac{dl}{d\theta} = \frac{d}{d\theta}\sum_{i=1}^{N}\left[x_i \log\theta + (1-x_i)\log(1-\theta)\right]$$

$$\frac{dl}{d\theta} = \sum_{i=1}^{N}\left[\frac{x_i}{\theta} - \frac{1-x_i}{1-\theta}\right]$$

$$\frac{d^2 l}{d\theta^2} = \frac{d}{d\theta}\sum_{i=1}^{N}\left[\frac{x_i}{\theta} - \frac{(1-x_i)}{(1-\theta)}\right] = \sum_{i=1}^{N}\left[-\frac{x_i}{\theta^2} - \frac{(-1)(1-x_i)(-1)}{(1-\theta)^2}\right]$$

$$\frac{d^2 l}{d\theta^2} = \sum_{i=1}^{N}\left[-\frac{x_i}{\theta^2} - \frac{1-x_i}{(1-\theta)^2}\right]$$

Letting the sum of x_i over all i equal x_{tot}, setting the first derivative equal to 0 and solving for θ:

$$\sum_{i=1}^{N}\left[\frac{x_i}{\theta} - \frac{1-x_i}{1-\theta}\right] = 0$$

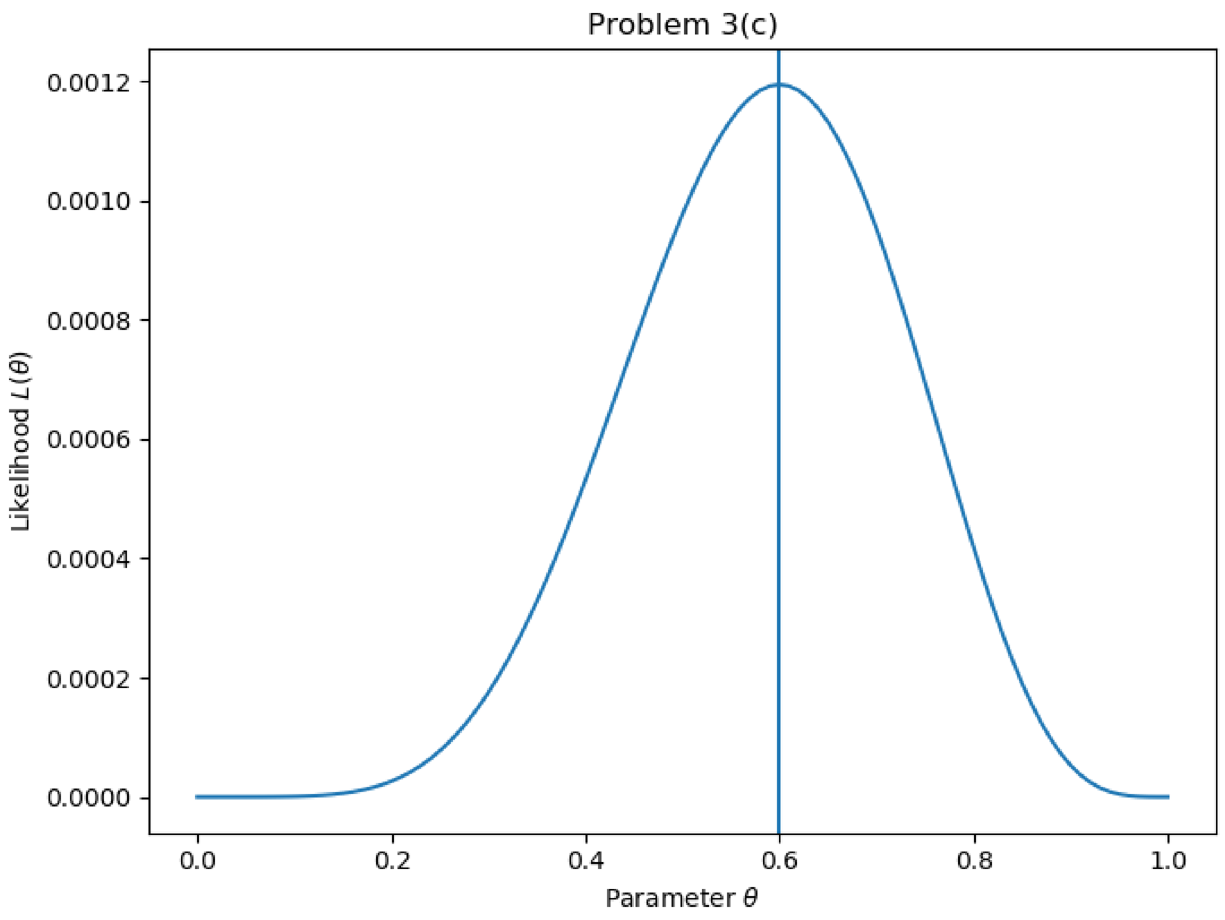
$$\sum_{i=1}^{N}\frac{x_i}{\theta} = \sum_{i=1}^{N}\frac{1-x_i}{1-\theta}$$

$$\frac{x_{\text{tot}}}{\theta} = \frac{N - x_{\text{tot}}}{1-\theta}$$

$$x_{\text{tot}} = N\theta$$

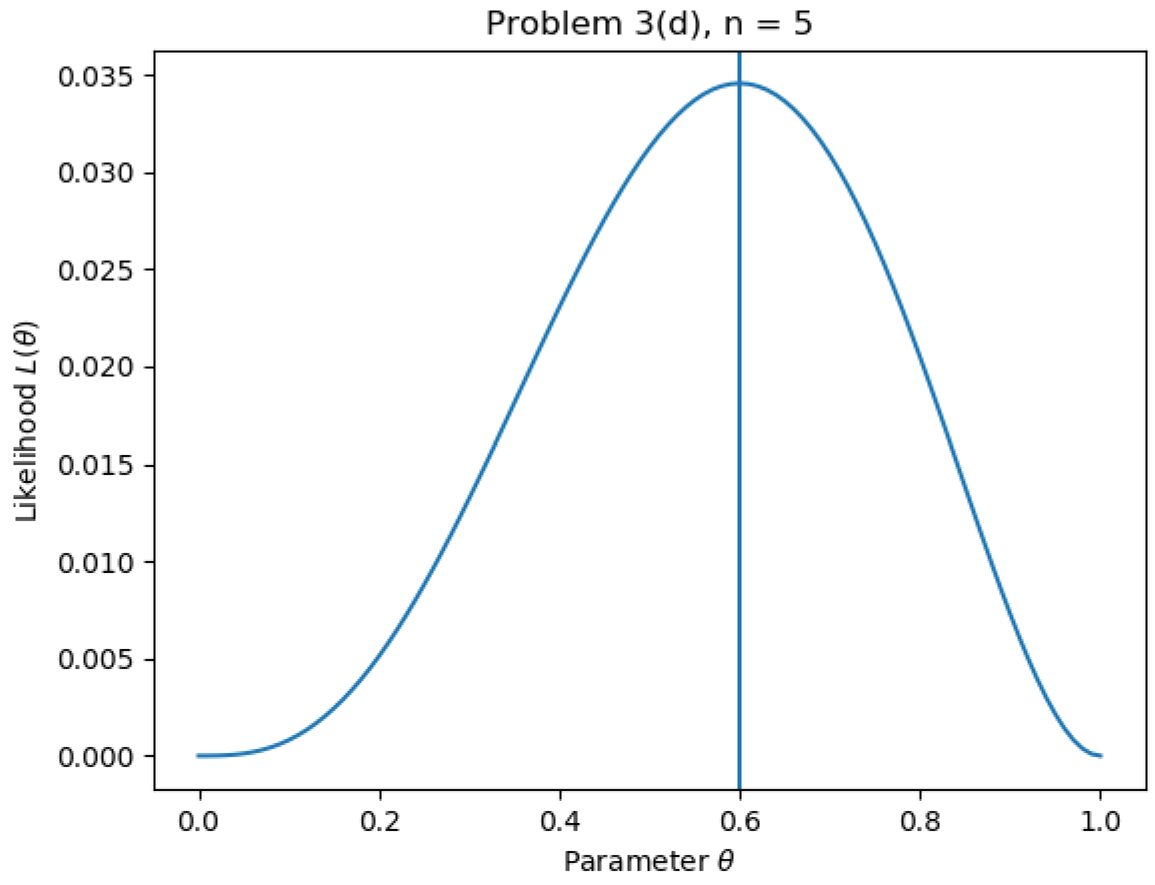
$$\theta = \overline{x}$$

(c)



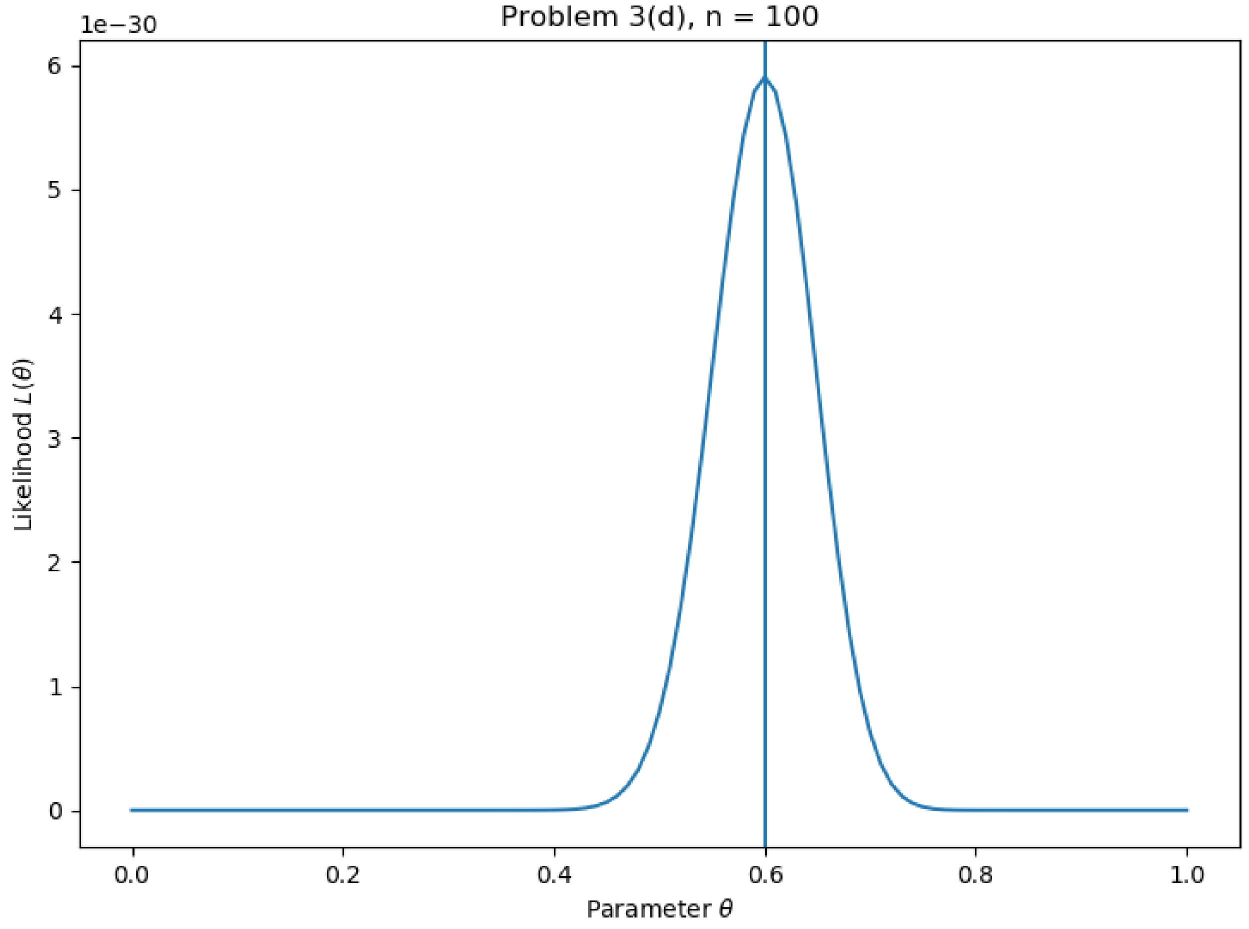
The value of θ that seems to maximize the likelihood is 0.6, which agrees with what I obtained using the result of 3(b).

(d) For $n = 5$, with three 1s and two 0s:



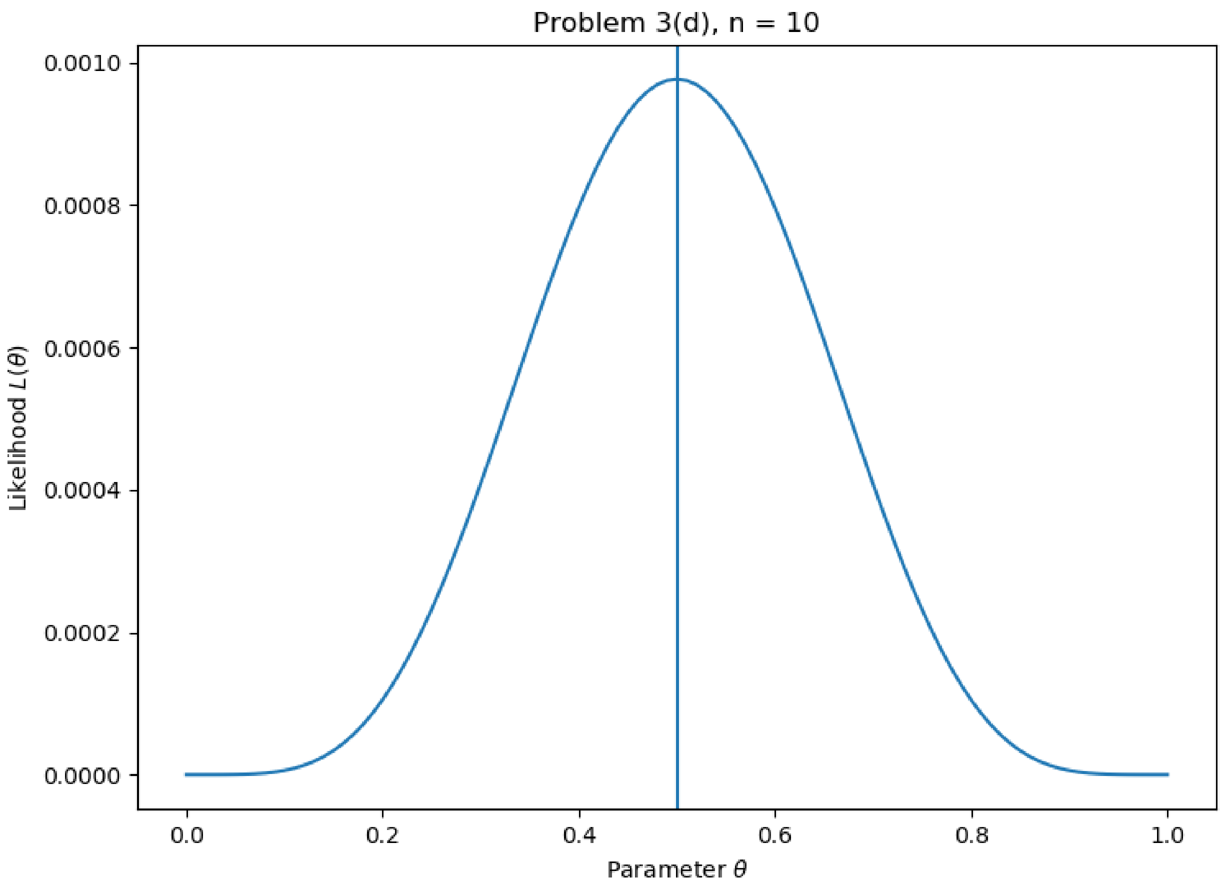
The graph is centered at $\theta = 0.6$, which is the same as the graph in 3(c), and it has the same bell curve shape. However, due to n being smaller (5 versus 10 for 3(c)), the standard deviation (spread) of the results is higher. Graphically, the "bell curve" is wider. Also, compared to $n = 10$, the maximum likelihood value is higher (about 0.035 versus about 0.0012 from 3(c)).

(d, continued)
For $n = 100$, with sixty 1s and forty 0s:



This graph is centered at $\theta = 0.6$, which is the same as the two previous graphs, and it still has the bell curve shape, but due to n increasing to 100, the width of the bell curve is much less (meaning $L(\theta)$ has a lower standard deviation). Also, the maximum likelihood value is much lower this time, on the order of 10^{-30}, compared to 0.035 for $n = 5$ and 0.0012 for $n = 10$.

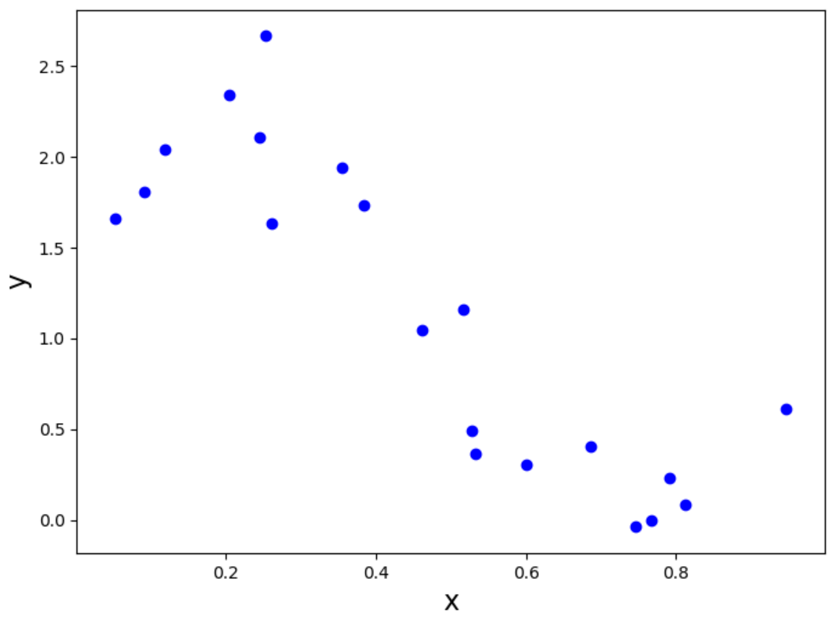
(d, continued)
For n = 10, with five 1s and five 0s:



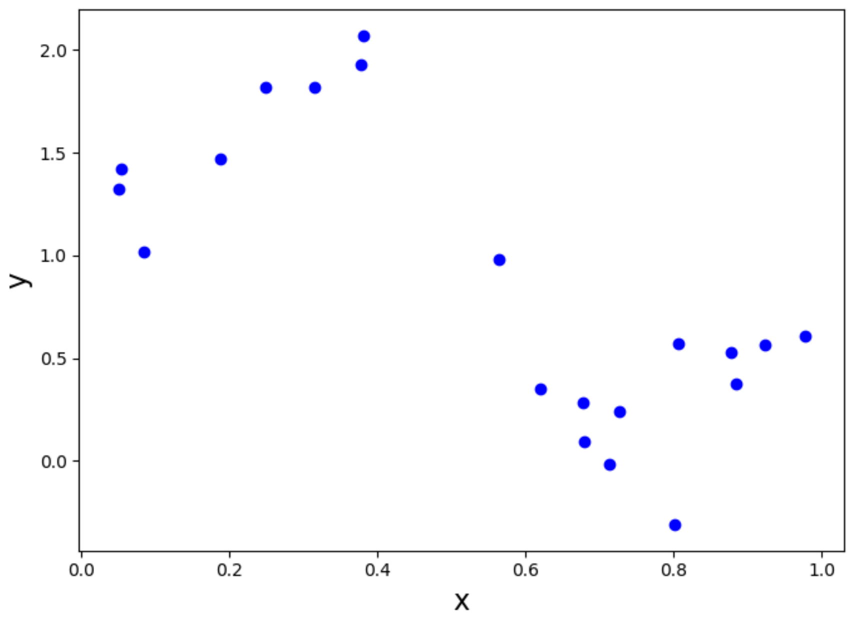
This time, unlike the other 3 preceding graphs, this one is centered at θ = 0.5, but it still has the same bell curve shape as all the other graphs. The spread seems to be roughly the same as the graph in 3(c). The maximum likelihood value is roughly the same in the graph for 3(c): this one is roughly 0.0010, while for 3(c) it is 0.0012.

Concluding remarks: looks like n affects standard deviation: as n increases, standard deviation decreases. The graph is centered about the proportion of ones in the data set. The tendency of the curve to look like the normal curve as n increases is due to the central limit theorem.

4. (a)



Above: Plotted training data.
Below: Plotted test data.



The top figure contains the training data and the bottom figure contains the test data. From the graphs, we can see that it looks like the data does not follow a linear pattern. Instead, a higher order polynomial or perhaps even a sine function would work better. Linear regression will not be very effective in this case.

(b) I implemented the specified code in the `generate_polynomial_features(...)` function.

(c) I implemented the specified code in the `predict(...)` function.

(d) Running the test code from the first bullet, I got 40.234, as expected. I then followed the instructions in the second bullet.

Running the training data through `fit_GD(...)`, I got the following results for different step sizes η:

Step Size, η	θ_0	θ_1	Number of Iterations
10^{-4}	2.27045	–2.46068	10000
10^{-3}	2.44641	–2.81635	7021
10^{-2}	2.44641	–2.81635	765
0.0407	-9.40471×10^{18}	-4.65229×10^{18}	10000

The results show that for a step size of around 10^{-2} does the best, and that 10^{-3} is a bit small but still large enough to converge within 10,000 iterations. However, when the step size is 10^{-4}, it is too small and takes too long to converge, as it reaches the 10,000 iterations limit. On the other hand, 0.0407 is too large of a step size, as it also reaches the 10,000 iterations limit but the coefficients are very different from the other step sizes and very large, indicating that the line is very likely to be incorrect. The step size is too large for the algorithm to converge. For the coefficients, it seems that the algorithm causes the coefficients to converge to roughly $\theta_0 = 2.44641$ and $\theta_1 = -2.81635$, with the results for $\eta = 10^{-3}$ being very close to the results for $\eta = 10^{-2}$. For $\eta = 10^{-4}$, it started making progress towards the values reported for $\eta = 10^{-2}$, but the step size was too small for a limit of 10,000 iterations, and so it wasn't able to reach the final values. And as previously stated, for $\eta = 0.0407$, the step size was too big, and so it never converged and thus reported extremely large values for the coefficients.

(e) The closed form solution is $\theta_0 = 2.446407$ and $\theta_1 = -2.816353$. To compare the performance of the exact solution to gradient descent, I reran the gradient descent, and also measured the cost and execution time.

Step Size, η	θ_0	θ_1	Number of Iterations	Execution Time (s)	Cost
10^{-4}	2.27045	–2.46068	10000	0.628	4.08674
10^{-3}	2.44641	–2.81635	7021	0.397	3.91258
10^{-2}	2.44641	–2.81635	765	0.021	3.91258
0.0407	-9.40471×10^{18}	-4.65229×10^{18}	10000	0.414	2.71092×10^{39}

Exact solution: run time of 0.00125 s (using `time.clock()`), cost of 3.912576.

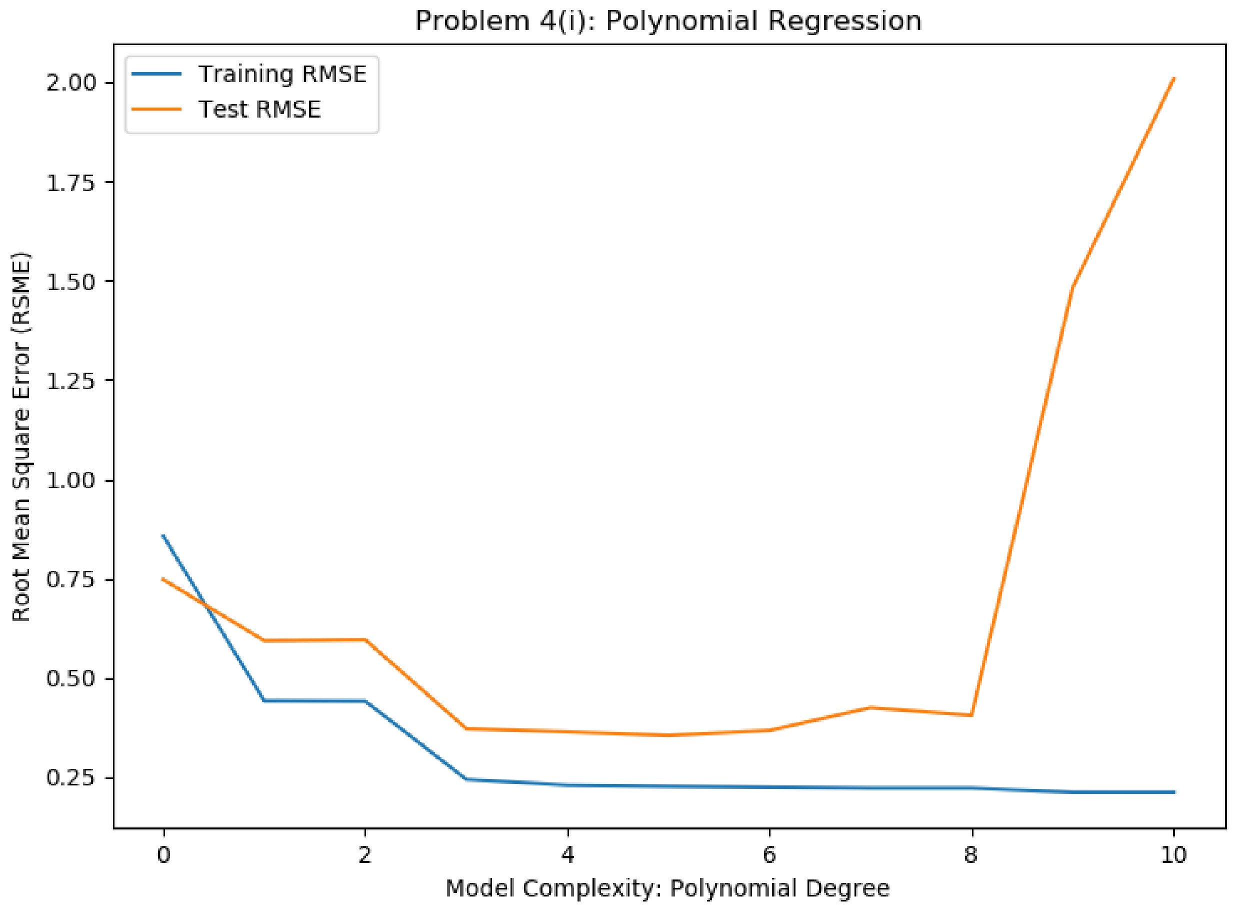
The coefficients and cost are nearly the same as those obtained by using gradient descent for step sizes of 0.01 and 0.001, indicating that gradient descent worked well. The runtime for the exact solution is significantly faster, and I suspect this is because the data set is relatively small, so even though the runtime of matrix inversion and multiplication is expensive, the data set is small enough that it does not matter. If there were more features for each data point and/or more data points, the runtime would dramatically increase.

(f) Using this iterative step size, the runtime decreases to 0.068 s, with the algorithm converging in 1780 iterations to values of θ_0 = 2.44641 and θ_1 = –2.81635, essentially the same as the results in part (d) (note: rounded to 5 decimal places). The cost, 3.91258 is the same as previous results too. As the iteration number increases, the step size decreases, so thus, at the start the algorithm quickly moves towards the minimum, and as it gets closer, it decreases its step size.

(g) I updated the `generate_polynomial_features(...)` as directed.

(h) We would prefer the root mean square error (RMSE) over the cost function because RMSE is a normalized measure of the accuracy of the prediction: it is a measure of the standard deviation of the target values about the value predicted by regression. Because RMSE involves division by the number of data points, it eliminates the size of a data set as a factor affecting the magnitude of error.

(i)



The minimum test error, 0.35514, occurs with a polynomial of degree 5. Underfitting can be seen on the left, where both the test and training errors are similarly high (in the neighborhood of 0.75 to 1). Then, as the degree of the polynomial increases (the model's complexity), the training error monotonically decreases, while the test error first decreases, and then suddenly increases at the end. The test error reaches its minimum at degree 5, as mentioned above, and the sudden increase in error as the degree becomes high while training error is low is a sign of overfitting.