EMORY UNIVERSITY

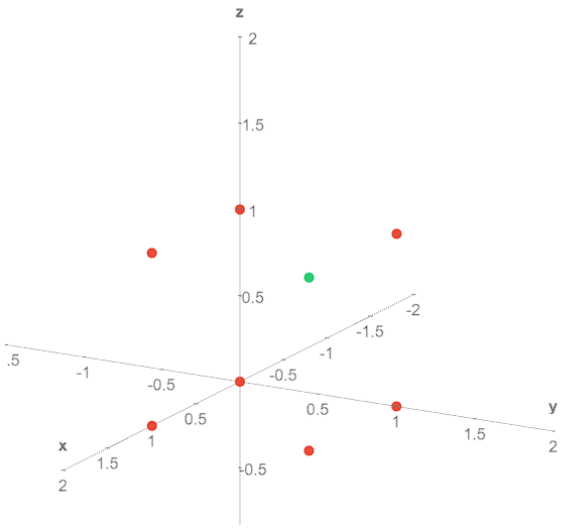
Homework 2

Student name: *Lance Ding*

Course: *CS334* – Professor: *Dr. Shengpu Tang*
Due date: *9/2024*

1. Decision Boundaries

(a) (i). It is impossible to learn such a $\vec{\theta}$, because the point $(0, 0, 0)$ will always be misclassified due to points on the decision boundary being treated as misclassified. Even without this rule, it is still impossible to learn such a $\vec{\theta}$.



The domain of our function can be captured by a cube with side length 1, with one corner at $(0, 0, 0)$ and the opposite corner at $(1, 1, 1)$. If we are not allowed a nonzero offset, then our decision boundary must pass through the origin, and must separate the corner at $(1, 1, 1)$ from all other points. This is clearly impossible.

(a) (ii). If a nonzero offset is allowed, then we could slice just the corner we want off. For example, the following parameters will correctly classify our points according to the logical AND operator:

$$\vec{\theta} = [1, 1, 1]^T$$

$$b = \frac{3}{2}$$

(b) (i). Let $r = \frac{1+\sqrt{2}}{2}$ be the radius of a circle centered at the origin. Points inside this circle should be classified as negative, and points outside should be classified as positive. In effect:

$$x^2 + y^2 < \left(\frac{1+\sqrt{2}}{2}\right)^2 \rightarrow \text{negative}$$

$$x^2 + y^2 > \left(\frac{1+\sqrt{2}}{2}\right)^2 \rightarrow \text{positive}$$

(b) (ii). This is impossible because the line has to go through the origin, and will always misclassify points. We would require a nonzero offset to achieve this with a straight line as a decision boundary.

(b) (iii). A square with a center at $(-1, 1)$ and side length $s = 1.5$ would work, with points on the inside being negative and points on the outside being positive.

(b) (iv). A square centered at the origin with $\alpha = \frac{\pi}{4}$ radians of rotation and a radius of $r = \frac{3}{2}$ - a side length of $s = \frac{3}{2}/\sqrt{2} = \frac{3}{2\sqrt{2}} = \frac{3\sqrt{2}}{4}$ would work. The inside of the square would classify as negative, and the outside would classify as positive.

2. Perceptron Algorithm with Offset

(a). Since $\vec{\theta} = \vec{0}$ initially, and each misclassification adds $y^{(i)}\vec{x}^{(i)}$ to θ, we get:

$$\begin{aligned}\vec{\theta} &= \sum_{i}^{N} \alpha_i y^{(i)} \vec{x}^{(i)} \\ &= (\vec{\alpha} \odot \vec{y}) \cdot \mathbf{X}\end{aligned}$$

where $\odot$ is the element-wise multiplication symbol, and $\mathbf{X}$ is the matrix whose i^{th} row is $\vec{x}^{(i)}$.

Since $b = 0$ initially, and each misclassification adds one $y^{(i)}$ to b, we get:

$$\begin{aligned}b &= \sum_{i}^{N} \alpha_i y^{(i)} \\ &= \vec{\alpha} \cdot \vec{y}\end{aligned}$$

(b). General formula for minimum (orthogonal) distance between a point and a plane:

$$d = \frac{\vec{\theta} \cdot \vec{x} + b}{||\vec{\theta}||}$$

where $\vec{\theta}$ is a plane normal vector, $\vec{x}_0$ is the point we are concerned with, and b is the offset. Since we want the distance from the origin, $\vec{x}_0 = \vec{0}$.

$$\therefore d = \frac{\vec{\theta} \cdot \vec{0} + b}{||\vec{\theta}||} = \frac{0 + b}{||\vec{\theta}||} = \frac{b}{||\vec{\theta}||}$$

(c). See code submission

(d). After running the code, we obtain:

$$\begin{aligned}\vec{\theta} &= [4, 0] \\ b &= -6\end{aligned}$$

Using the expressions derived from **(a)** and the objects returned by the **perceptron()** method, we arrive at the same $\vec{\theta}$ and b. See code for details.

(e). The order that the points are presented to the algorithm does not affect whether or not the algorithm will converge. However, it may affect the speed of convergence/number of errors it makes before it converges due to the different paths that it may take along the loss surface. This can be easily verified by shuffling the points. See code for an example utilizing sklearn's shuffle function.

3.1 Linear Regression - Optimization Method

(a). See code submission

(b). See code submission

(c). See code submission

(d). The table below has been filled out with the values from running the code written above, taking the output from the console and directly pasting it into a spreadsheet. Note that runtimes of 0 are not actually instantaneous - rather they are so short that the timing library was not granular enough for it.

	A	B	C	D	E	F
1	Algorithm	eta	theta 0	theta 1	# Iterations	Runtime (s)
2	GD	10^-4	0.35654059	0.02205277	28902	1.203125
3	GD	10^-3	0.35739911	0.01984747	3713	0.15625
4	GD	10^-2	0.35464116	0.0176296	349	0.015625
5	GD	10^-1	0.31333059	-0.00974265	100	0
6	SGD	10^-4	0.35655012	0.02210486	28815	1.296875
7	SGD	10^-3	0.35634314	0.02254004	2762	0.125
8	SGD	10^-2	0.36007198	0.01855923	1121	0.046875
9	SGD	10^-1	0.42768374	0.04551579	844625	37.8125
10	Closed form	-	0.35820909	0.01915174	-	0
11						
12						

(e). GD vs SGD is interesting. Looking at the number of iterations, we observe a comparable number with the learning rates of 10^{-4}. SGD takes significantly less iterations than GD with a learning rate of 10^{-3}, and GD takes significantly less iterations than SGD with a learning rate of 10^{-2} and 10^{-1}. Since the runtime of these algorithms are directly proportional to the number of iterations, we observe a similar pattern in the runtime. At the three smaller learning rates, GD and SGD converged roughly to the same point, with $\theta_0 \approx 0.355 \pm 0.002$ and $\theta_1 \approx 0.020 \pm 0.004$. At the largest learning rate $\eta = 10^{-1}$, both GD and SGD converged to a $\vec{\theta}$ that was not close to the previous results. This is probably partly due to the large learning rate making the weights overshoot with each update, as indicated by the extremely large number of iterations the SGD algorithm took. Overall, the agreement of the $\vec{\theta}$s resulting from the smaller learning rates suggest that smaller learning rates are better for ensuring convergence, but as seen by the runtimes of the algorithms, one likely needs to find a good balance between ensuring convergence accuracy and convergence speed.

(f). The closed-form solution was easily orders of magnitude faster than any of the SGD algorithms. The results obtained from SGD with $\eta = 10^{-2}$ was the closest to the closed-form solution.

(g). Let the learning rate η be a function of k, the number of iterations be defined as follows:

$$\eta(k) = \frac{1}{1+k}$$

This heuristic satisfies the **Robbins-Monro** conditions mentioned in earlier lectures:

$$(1) \sum_{k=0}^{\infty} \eta(k) = \sum_{k=0}^{\infty} \frac{1}{1+k} = \sum_{k=1}^{\infty} \frac{1}{n} = \infty$$

$$(2) \sum_{k=0}^{\infty} (\eta(k))^2 = \sum_{k=0}^{\infty} \left(\frac{1}{1+k}\right)^2 = \sum_{k=0}^{\infty} \frac{1}{(1+k)^2} = \sum_{k=1}^{\infty} \frac{1}{k^2} < \infty \text{ by p-series test, p>1}$$

With this adaptive learning rate, the algorithm's performance is as follows:

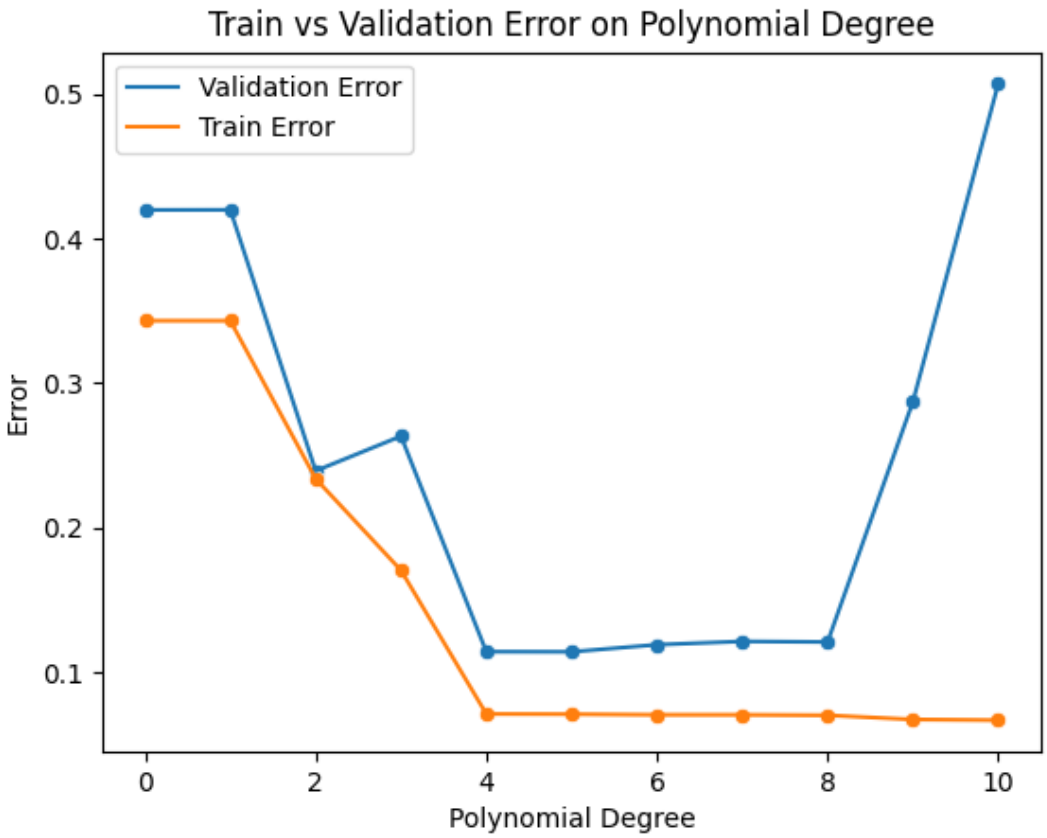
Algorithm	eta	theta 0	theta 1	# Iterations	Runtime (s)
SGD	adaptive	0.35790304	0.01922669	311	0.015625

The coefficients produced with the adaptive learning rate very closely matches the closed-form solution - closer than any previous $\vec{\theta}$, in fact. Additionally, it converges very quickly, taking the second least amount of iterations and hence runtime to finish out of all the GD and SGD runs. This demonstrates that a good adaptive learning rate can greatly boost an algorithm's efficiency and accuracy.

3.2 Linear Regression - Overfitting & Regularization

(a). See code submission

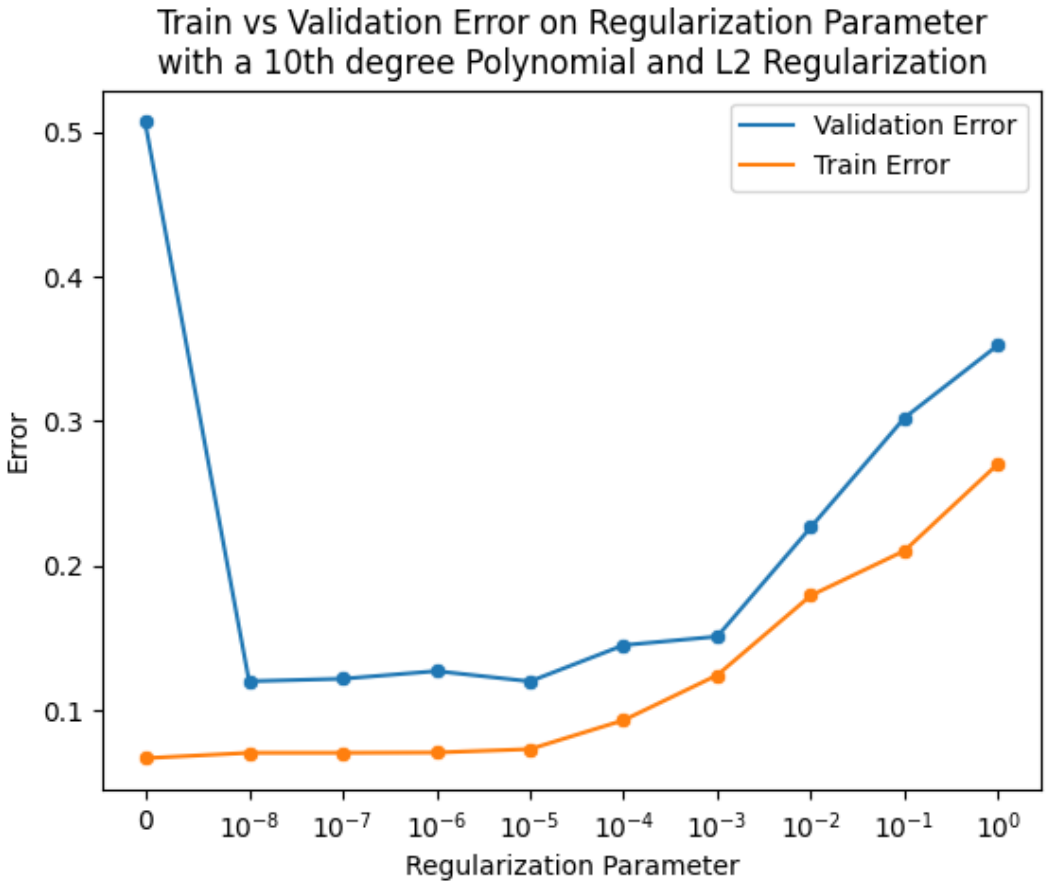
(b).



(c). It looks like polynomials of degree 4-8 all display similar levels of loss, with relatively equal levels of generalization error, as indicated by the near-constant vertical distance between corresponding points on the plot. I would prefer a simpler model due to Occam's razor, so I would go with the 4th degree polynomial.

(d). See code submission

(e).



(f). For a 10th degree polynomial, it looks like a regularization parameter of $\lambda = 10^{-8}$ works best, as indicated by the low error both in training and validation datasets, as well as the small generalization gap, as indicated by the vertical distance between the two lines on the plot.

(g). While the effect of regularization is shown to be good as per **(f)**, it is unclear from the experiments that we have conducted so far how the effect of different regularization strengths on different degree polynomials may differ. For a better idea, I might conduct the experiment from **(e)** with a 4th or maybe 8th degree polynomial to see what's best. Given that the search space is small and the cost of computation is low for our case, I could even try all the combinations of the polynomial degrees and regularization parameters and select the one with the lowest errors and generalization gap.

3.3 Weighted Linear Regression

(a). It may be useful to weigh examples differently during training if a population is being underrepresented. Under-representation may cause statistical bias to permeate into our model. Another case where employing weighted regression could be if we know certain sources are more "trustworthy" or have less noise than others.

(b). We look at $J(\vec{\theta})$'s components separately.

The first part of $J(\vec{\theta})$, looks very similar to $(X\vec{\theta} - \vec{y})^T \cdot (X\vec{\theta} - \vec{y})$, but each term in the sum (during the dot product) has an extra scalar $w^{(i)}$. We can easily attach the w scalars to their corresponding entries by storing them in a diagonal matrix, since multiplying a vector by a diagonal matrix has the effect of scaling each term of the vector by the corresponding entry on the diagonal. Therefore, something to the effect of:

$$(X\vec{\theta} - \vec{y})^T \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y})$$

where **W** is:

$$\begin{bmatrix} w^{(1)} & 0 & ... & 0 & 0 \\ 0 & w^{(2)} & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & w^{(n-1)} & 0 \\ 0 & 0 & ... & 0 & w^{(n)} \end{bmatrix}$$

would achieve the desired effect of multiplying the entries with the corresponding weights without changing the dimensions of any vectors.

Trivially, the second part of $J(\vec{\theta})$, $\lambda||\vec{\theta}||_2^2 = \lambda\vec{\theta}^T \cdot \vec{\theta}$.

Therefore, $J(\vec{\theta}) = (X\vec{\theta} - \vec{y})^T \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y}) + \lambda\vec{\theta}^T \cdot \vec{\theta}$.

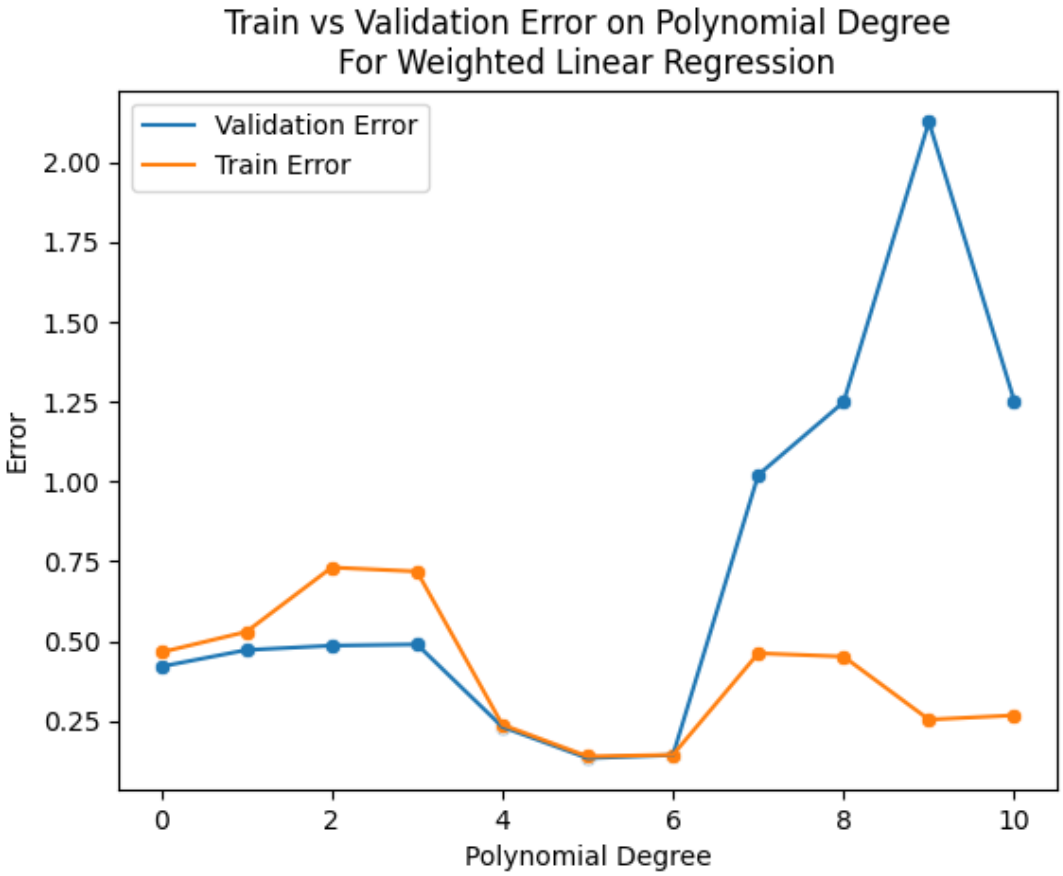
(c).

$$
\begin{aligned}
\nabla_{\vec{\theta}} J(\vec{\theta}) &= \nabla_{\vec{\theta}} \left[(X\vec{\theta} - \vec{y})^T \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y}) + \lambda \vec{\theta}^T \cdot \vec{\theta} \right] \\
&= \nabla_{\vec{\theta}} \left[(X\vec{\theta} - \vec{y})^T \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y}) \right] + \nabla_{\vec{\theta}} \left[\lambda \vec{\theta}^T \cdot \vec{\theta} \right] \\
&= \nabla_{\vec{\theta}} \left[(X\vec{\theta} - \vec{y})^T \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y}) \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[(\vec{\theta}^T X^T - \vec{y}^T) \cdot \mathbf{W} \cdot (X\vec{\theta} - \vec{y}) \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[(\vec{\theta}^T X^T \mathbf{W} - \vec{y}^T \mathbf{W}) \cdot (X\vec{\theta} - \vec{y}) \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[\vec{\theta}^T X^T \mathbf{W} X\vec{\theta} - \vec{y}^T \mathbf{W} X\vec{\theta} - \vec{\theta}^T X^T \mathbf{W}\vec{y} + \vec{y}^T \mathbf{W}\vec{y} \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[\vec{\theta}^T X^T \mathbf{W} X\vec{\theta} - \vec{y}^T \mathbf{W} X\vec{\theta} - \vec{\theta}^T X^T \mathbf{W}\vec{y} \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[\vec{\theta}^T X^T \mathbf{W} X\vec{\theta} - \vec{y}^T \mathbf{W}^T X^{T^T} \vec{\theta}^{T^T} \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[\vec{\theta}^T X^T \mathbf{W} X\vec{\theta} - \vec{y}^T \mathbf{W} X\vec{\theta} - \vec{y}^T \mathbf{W} X\vec{\theta} \right] + 2\lambda\vec{\theta} \\
&= \nabla_{\vec{\theta}} \left[\vec{\theta}^T X^T \mathbf{W} X\vec{\theta} - 2\vec{y}^T \mathbf{W} X\vec{\theta} \right] + 2\lambda\vec{\theta} \\
&= 2X^T \mathbf{W} X\vec{\theta} - 2\vec{y}^T \mathbf{W} X + 2\lambda\vec{\theta} \\
&= 2X^T \mathbf{W} X\vec{\theta} - 2X^T \mathbf{W}^T \vec{y}^{T^T} + 2\lambda\vec{\theta} \\
&= 2X^T \mathbf{W} X\vec{\theta} - 2X^T \mathbf{W}\vec{y} + 2\lambda\vec{\theta} \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\therefore X^T \mathbf{W} X\vec{\theta} + \lambda\vec{\theta} &= X^T \mathbf{W}\vec{y} \\
\left(X^T \mathbf{W} X + \lambda \mathbf{I}_d \right) \vec{\theta} &= X^T \mathbf{W}\vec{y} \\
\left(X^T \mathbf{W} X + \lambda \mathbf{I}_d \right)^{-1} \left(X^T \mathbf{W} X + \lambda \mathbf{I}_d \right) \vec{\theta} &= \left(X^T \mathbf{W} X + \lambda \mathbf{I}_d \right)^{-1} X^T \mathbf{W}\vec{y} \\
\therefore \vec{\theta}^* &= \left(X^T \mathbf{W} X + \lambda \mathbf{I}_d \right)^{-1} X^T \mathbf{W}\vec{y} \qquad \square
\end{aligned}
$$

(d). See code submission

(e).



As the above plot indicates, the best polynomial for our dataset and weights is either a 5th or a 6th degree polynomial. Although this falls under the range of degrees that I suggested in §3.2, I do not expect this to hold in general. Additionally, we see a different set of error curves and trends. This is to be expected, since adding weights to the regression effectively gives us a new dataset (albeit in the span of the original).