HW3 Report

Section 1

Q1:

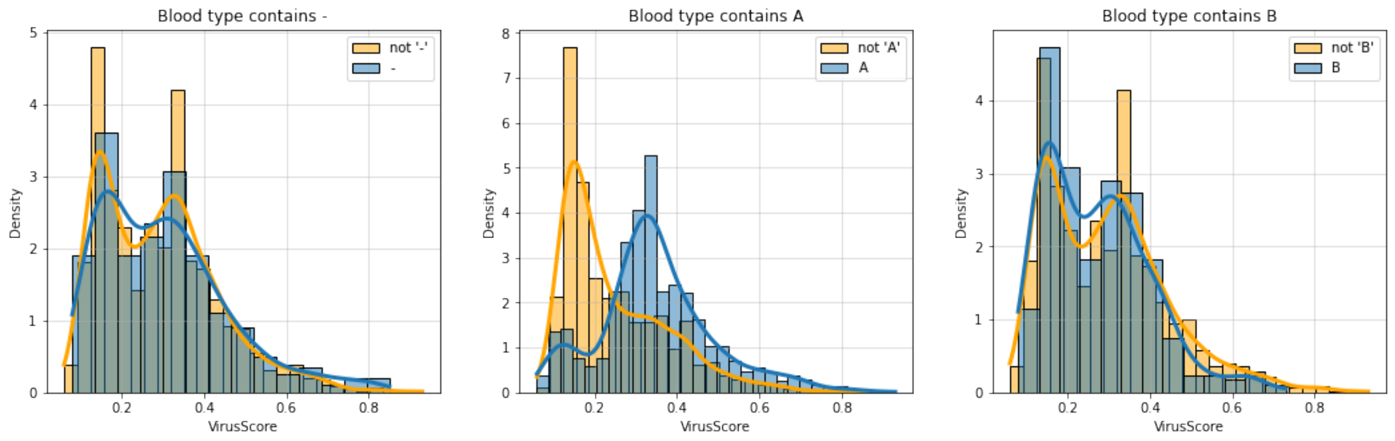

Figure 1: KDE plots of `VirusScore` conditioned on different conditions of `blood_type`

Q2:

In figure 1 in the plot of A versus not A, we observe that the groups of patients with and and without "A" in their blood types are mostly seperable along a boundary that is approximately the `VirusScore` of 0.225.

Therefore, the condition of contains/does not contain A would be most informative for learning `VirusScore`. As it turns out, we decided already in hw1 to create this feature.

Q3:

$$\frac{\partial}{\partial b}\mathcal{L}\left(\underline{\mathbf{w}},b\right) \overset{a}{=} \frac{\partial}{\partial b}\frac{1}{m}\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i+b-y_i\right)^2 \overset{b}{=} \frac{1}{m}\frac{\partial}{\partial b}\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i+b-y_i\right)^2 \overset{c}{=} \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}\left(\underline{w}^{\top}\underline{x}_i+b-y_i\right)^2 \overset{d}{=}$$

$$\overset{d}{=} \frac{1}{m}\sum_{i=1}^{m}2\left(\underline{w}^{\top}\underline{x}_i+b-y_i\right)\cdot(1) \overset{e}{=} \frac{2}{m}\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i+b-y_i\right) \overset{f}{=} \frac{2}{m}\left[mb+\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i-y_i\right)\right] \overset{g}{=}$$

$$\overset{g}{=} 2b+\frac{2}{m}\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i-y_i\right) \Rightarrow \frac{\partial}{\partial b}\mathcal{L}\left(\underline{\mathbf{w}},b\right) = 2b+\frac{2}{m}\sum_{i=1}^{m}\left(\underline{w}^{\top}\underline{x}_i-y_i\right)$$

a : Definition of $\mathcal{L}\left(\underline{\mathbf{w}},b\right)$
b : $\frac{1}{m}$ is scalar
c : Derivative of a sum is the sum of derivatives
d : Derivative of $\left(\underline{w}^{\top}\underline{x}_i-b-y_i\right)^2$w.r.t b
e : 2 is scalar
f : Sum of b
g : Removing b from the sum

Q4:

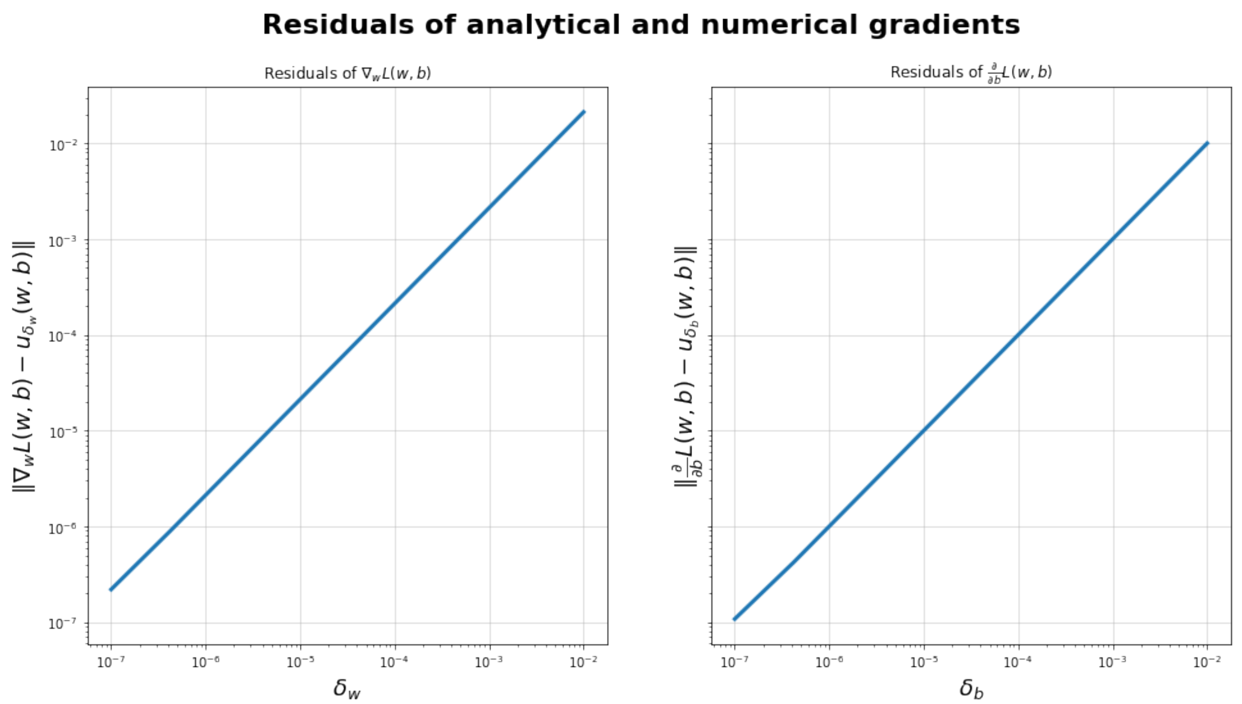

Figure 2: Plot of Residuals of analytical and numerical gradients

As we can see in figure 2, the difference between the analytic and numerical gradients increases in a monotonic fashion as the value of δ increases. This is logical, as δ is the differential size used in the definition of the numerical gradient, and therefore a smaller δ equates to a more precise estimation of the analytic gradient by the numerical gradient.

Q5:

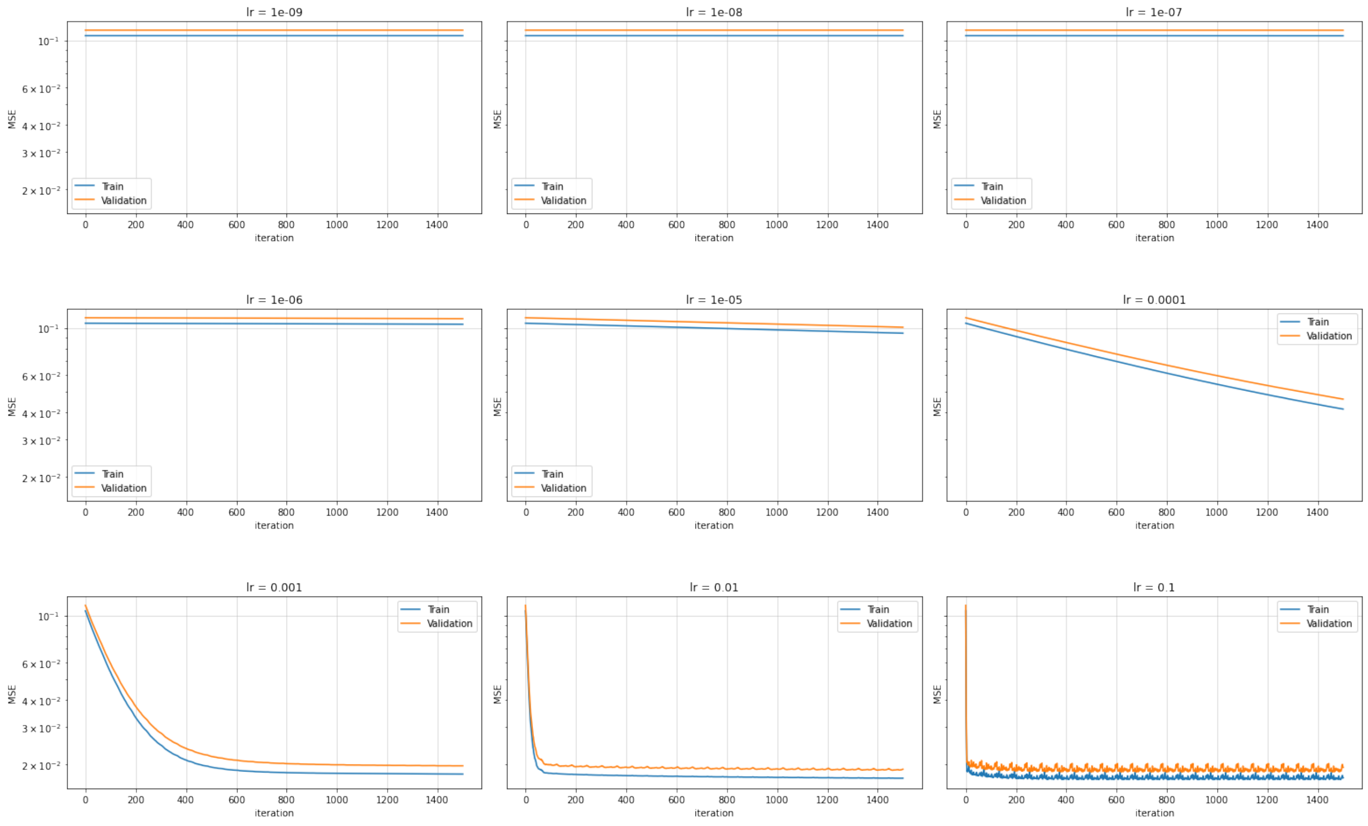

Figure 3: Graphs of Training and Validation Losses as Functions of Iteration Number for Different Learning Rates

We can see in figure 3 that for smaller lr, for those that converge, the convergence is at a higher loss for both the training and validation. This matches the theory, since if the lr is too small, the SGD algorithm is likely to converge to a sub-optimal solution. In addition we observe that for the lr equal to 0.1 the graphs do not converge for both training and validation losses and for lr equal to 0.01 the validation loss does not converge, which fits the theory that says that learning rates that are too high are likely to cause the SGD algorithm to take steps that are too large and thus repeatedly skip-over the optimal solution. This points to 0.001 as being the optimal lr, as both the validation and training losses converge to values that are substantially lower than the next smaller lr.

Q6:

Model	Section	Train MSE	Valid MSE
		Cross Validated	
Dummy	3	0.0204	0.0205

Figure 4: Training and Validation MSE Errors for Dummy Regressor

Section 4

Q7:

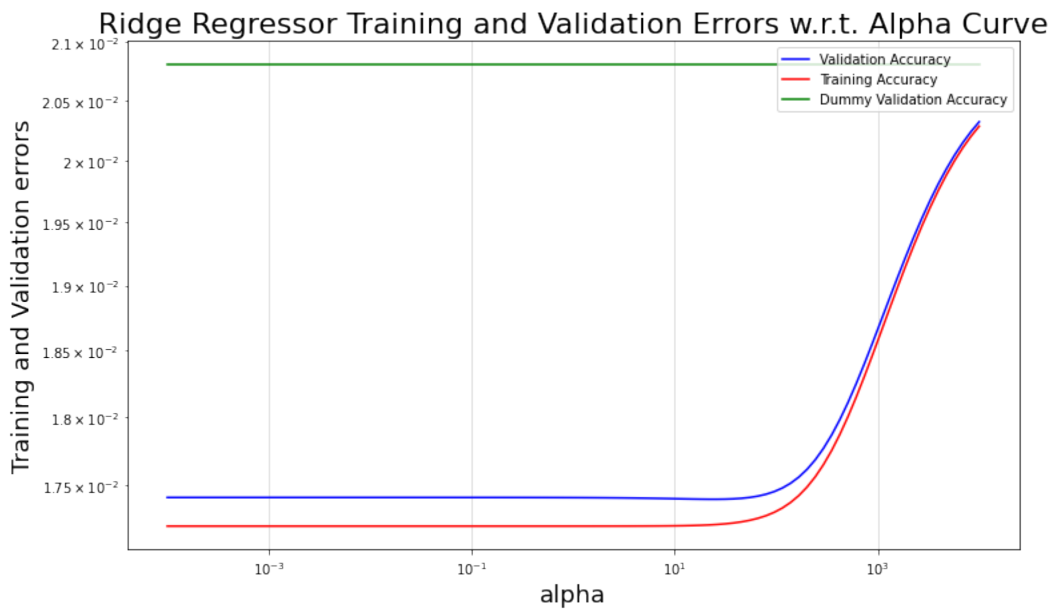

Figure 5: Graph of Training and Validation Errors of Ridge Regressor as a Function of `alpha`

Optimal alpha for validation: 25.950242113997373
Validation error for optimal alpha: 0.017402908351734445

Q8:

Model	Section	Train MSE	Valid MSE
		Cross Validated	
Dummy	3	0.0204	0.0205
Ridge linear	4	0.0172	0.0174

Figure 6: Training and Validation MSE Errors for Dummy and Ridge Regressors

Q9:

Feature	Coefficient
`blood_A_AB`	0.0970
`num_of_siblings`	0.0298
`household_income`	−0.0275
`PCR_02`	−0.0141
`PCR_01`	−0.0113

Figure 7: Five Features From Ridge Regressor With Largest Coefficients in Terms of Absolute Value, From Largest to Smallest

Q10:

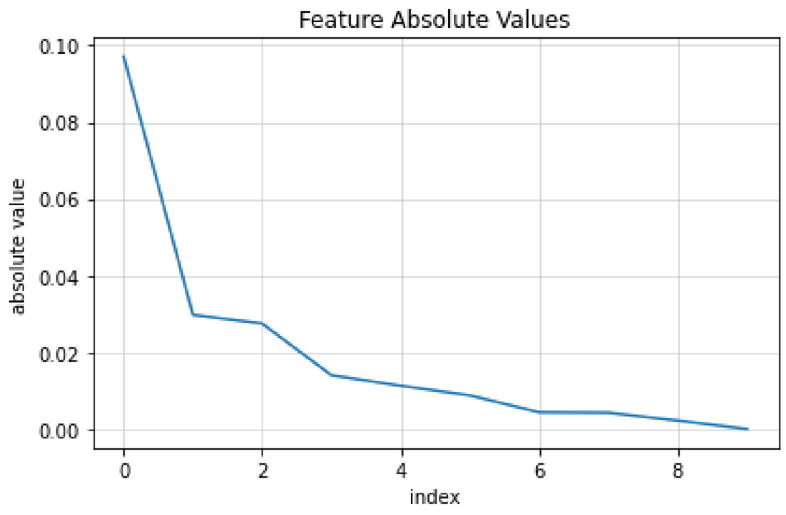

Figure 8: Graph of Absolute Values of Learned Coefficients of Ridge Regressor

Section 5

Q11:

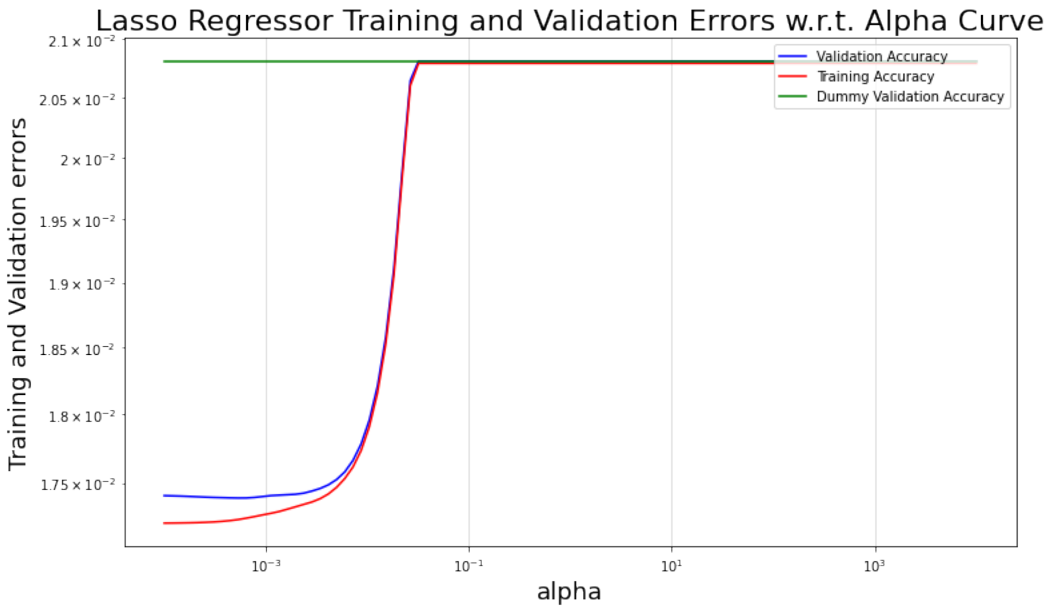

Figure 9: Graph of Training and Validation Errors of Lasso Regressor as a Function of `alpha`

Optimal alpha for validation: 0.0005336699231206312
Validation error for optimal alpha: 0.01739141826673015

Q12:

Yes, the graph for tunning the λ of the lasso regressor as can be seen in figure 9 is different from the graph obtained for the ridge regressor in figure 5 in a number of aspects.

Firstly, the steepness of the upwards curve is much more abrupt and significant in the case of the lasso regressor than it is for the ridge. This can be explained by the way in which lasso differs than ridge: Whereas a ridge regressor utilizes ℓ^2 regularization, lasso is implemented with ℓ^1. This means that as λ (`alpha`) is increased, the weight coefficients for the two regressors decrease in different manners. In the case of ridge, the coefficients experience weight decay, meaning they gradually tend to zero, whereas lasso tends to perform variable selection, meaning that the coefficients are more abruptly forced to zero one-by-one. This is reflected in the graphs, as the abruptness seen in figure 9 could be explained by a certain significant coefficient abruptly forced to zero, whereas in the ridge graph of figure 5, coefficients are more gradually lowered and therefore the curve is more gradual.

Furthermore, the upwards trend in the graph of lasso begins at a much lower `alpha` approximately equal to 0.025 than it does for ridge (approximately `alpha` equal to 70). This can be again explained by the behavior of ℓ^1 as opposed to ℓ^2: The decision of the lasso regressor to decrease a significant coefficient would be more abrupt and harsher than in the case of ridge, and therefore its effect on the accuracy would be seen almost immediately, and hence the earlier upwards trend in the lasso graph.

Lastly, the optimal `alpha` for minimal validation error in thew case of lasso, equal to 0.0005, is much smaller than that of ridge (25.95).

Q13:

Model	Section	Train MSE	Valid MSE
		Cross Validated	
Dummy	3	0.0204	0.0205
Ridge linear	4	0.0172	0.0174
Lasso linear	5	0.0172	0.0173

Figure 10: Training and Validation MSE Errors for Dummy, Ridge, and Lasso Regressors

Q14:

Feature	Coefficient
`blood_A_AB`	0.0993
`num_of_siblings`	0.0297
`household_income`	-0.0223
`PCR_02`	-0.0064
`PCR_10`	0.0038

Figure 11: Five Features From Lasso Regressor With Largest Coefficients in Terms of Absolute Value, From Largest to Smallest

Q15:

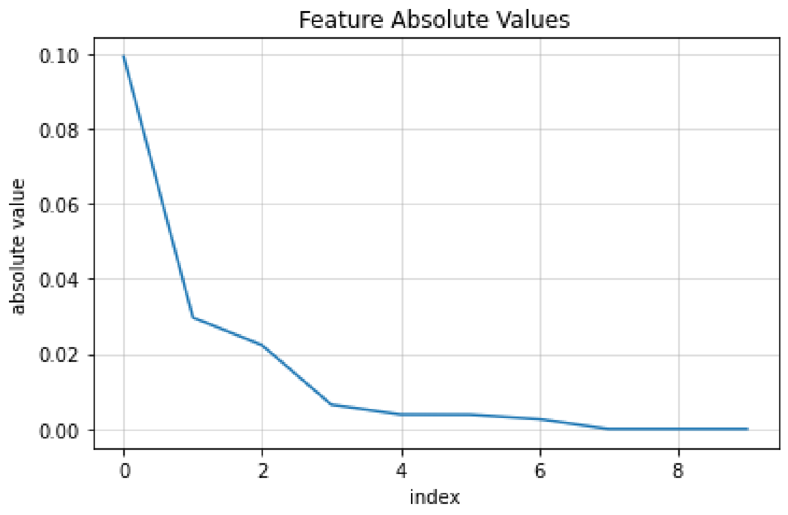

Figure 12: Graph of Absolute Values of Learned Coefficients of Lasso Regressor

In figure 12 we see see that the Lasso model sets absolute values for each parameter which are equal or lower than those set by the Ridge model (as seen in figure 8). From this we can conclude that the Ridge model causes weight decay with ℓ^2 regularization because it doesn't "dispose" within calculations the unimportant features but still gives it weight higher than 0. As for the Lasso model, we see that it causes variable selection with ℓ^1 regularization (which include sparse solutions) as it ignores unimportant feature and gives less weight than Ridge model to some features. Another behavior which indicates the differences between the models is that the Lasso features coefficient reaches 0 faster than the Ridge model.

Section 6

Q16:

When using a polynomial feature mapping, we can expect the training error to decrease and the validation error to increase. This is because the polynomial mapping will give more flexibility to the regression model to more closely try to fit the training data during training in a polynomial way, thereby leading to lower training error, but also causing overfitting and therefore leading to a higher validation error.

Q17:

Let $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d-1} \\ 1 \end{bmatrix}$ be a d-dimensional feature vector, and z be the binary feature from question 2. Let us then define the following 2nd degree polynomial mapping ϕ:

$$\phi(\underline{x}) = \begin{bmatrix} x_1 \cdot z \\ x_2 \cdot z \\ \vdots \\ x_{d-1} \cdot z \\ 1 \cdot z \\ x_1 \cdot (1-z) \\ x_2 \cdot (1-z) \\ \vdots \\ x_{d-1} \cdot (1-z) \\ 1 \cdot (1-z) \end{bmatrix}$$

Let us now define the linear model $h_{poly}(\underline{x}) = \underline{w^*}^T \phi(\underline{x})$, where $\underline{w^*} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d-1} \\ b_1 \\ w_{d+1} \\ w_{d+2} \\ \vdots \\ w_{2d-1} \\ b_2 \end{bmatrix}$ is a $2d$ dimensional weight vector.

Let us also define $\underline{w_1} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d-1} \\ b_1 \end{bmatrix}$ and define $\underline{w_2} = \begin{bmatrix} w_{d+1} \\ w_{d+2} \\ \vdots \\ w_{2d-1} \\ b_2 \end{bmatrix}$.

Now let us consider the result of h_{poly} applied on $\underline{x}$ where the feature z in $\underline{x}$ is equal to 1:

$$\begin{aligned} h_{poly}(x) &= \underline{w^*}^T \phi(\underline{x}) \\ &= w_1 \cdot x_1 \cdot z + \cdots + w_{d-1} \cdot x_{d-1} \cdot z + b_1 \cdot 1 \cdot z + w_{d+1} \cdot x_1 \cdot (1-z) + \cdots + w_{2d-1} \cdot x_{d-1} \cdot (1-z) + b_2 \cdot 1 \cdot (1-z) \\ &= w_1 \cdot x_1 + \cdots + w_{d-1} \cdot x_{d-1} + b_1 + 0 + \cdots + 0 + 0 \\ &= w_1 \cdot x_1 + \cdots + w_{d-1} \cdot x_{d-1} + b_1 \\ &= \underline{w_1}^T \underline{x} + b_1 \end{aligned}$$

Now let us consider the result of $h_{poly}(x)$ applied on x where the feature z in x is equal to 0:

$$\begin{aligned} h_{poly}(x) &= w{*}^T \phi(x) \\ &= w_1 \cdot x_1 \cdot z + \cdots + w_{d-1} \cdot x_{d-1} \cdot z + b_1 \cdot 1 \cdot z + w_{d+1} \cdot x_1 \cdot (1-z) + \cdots + w_{2d-1} \cdot x_{d-1} \cdot (1-z) + b_2 \cdot 1 \cdot (1-z) \\ &= 0 + \cdots + 0 + 0 + w_{d+1} \cdot x_1 + \cdots + w_{2d-1} \cdot x_{d-1} + b_2 \\ &= \underline{w_2}^T \underline{x} + b_2 \end{aligned}$$

We have shown that $h_{poly} = h_{multi}$. Therefore $h_{multi} = h_{poly} \in \mathcal{H}_{poly}$.

A 2nd degree polynomial model has more complex and flexible features and therefore mean it can more exactly model the training data, leading to lower training error but higher validation error due to overfitting. In the other hand, h_{multi} has less rich features and therefore is less likely to precisely fit the training data leading to higher training error but also mean it is more general and therefore more likely to have lower validation error.

Q18:

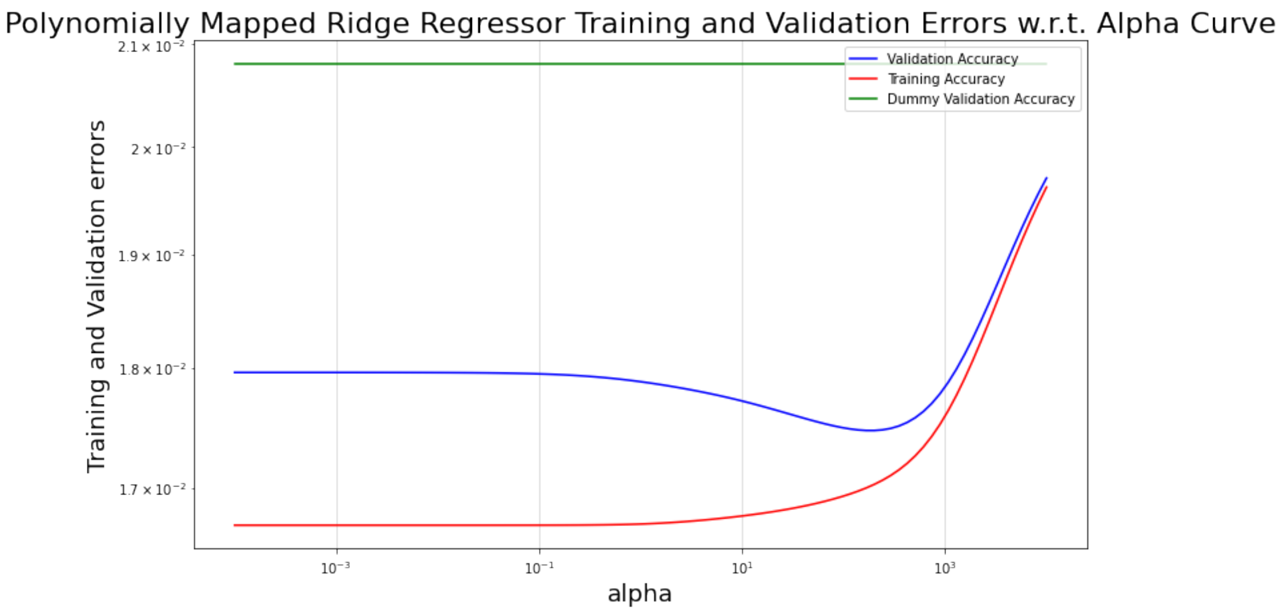

Figure 13: Graph of Training and Validation Errors of Polynomially-Mapped Ridge Regressor as a Function of `alpha`

Optimal alpha for validation: 200.923300256505
Validation error for optimal alpha: 0.01747015013288799

Q19:

Model	Section	Train MSE	Valid MSE
		Cross Validated	
Dummy	3	0.0204	0.0205
Ridge linear	4	0.0172	0.0174
Lasso linear	5	0.0172	0.0173
Ridge polynomial	6	0.0166	0.0174

Figure 14: Training and Validation MSE Errors for Dummy, Ridge, Lasso, and Polynomial Ridge Regressors

Section 7

Q20:

Model	Section	Train MSE	Valid MSE	Test MSE
		Cross Validated		Retrained
Dummy	3	0.0204	0.0205	0.0182
Ridge linear	4	0.0172	0.0174	0.0163
Lasso linear	5	0.0172	0.0173	0.0162
Ridge polynomial	6	0.0166	0.0174	0.0164

Figure 15: Training, Validation, and Test MSE Errors for Dummy, Ridge, Lasso, and Polynomial Ridge Regressors

As can be seen in figure 15, the lasso regressor performed the best on the test set, as it achieved the lowest MSE of 0.0162. On the other hand, the dummy regressor performed the worst, as expected.

Contrary to what is expected according to the theory, the test scores were better for all the models than their respective scores for both the training and validation sets, suggesting an occurrence of underfitting. This is likely due to the fact that the optimal values for *lambda* for the models in terms of validation accuracy where relatively large except for lasso, meaning that our models where more heavily regularized, and thus expected to have have lower variance and thus better fitting of unseen data, at the cost of lower accuracy in terms of training. Another possible explanation for this is that the models performed better once they were trained on a larger dataset. A third (more unlikely) explanation is that the test set formed by our seed contains data that happens to fit our trained models better than the actual training set.