PHP2650: Statistical Learning and Big Data

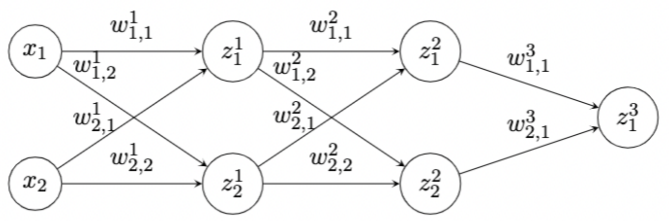
Assignment 5 - Supervised Machine Learning

Antonella Basso

May 6, 2022

Problem 1: Back-propagation and Estimation in Neural Networks

Consider the network below which takes an input with two values and has two hidden layers each with two nodes and ReLU activation functions along with a single output node with the identity function (that is it outputs a linear combination of the two previous node's values). Given a current set of weights w on each of the edges in the network, find the gradient by calculating the derivatives in the order that back-propagation would do so.



Solution

Given that we have a single continuous outcome, it follows that the appropriate cost and output functions for our generalized feedforward neural network will be the mean squared error (MSE) and identity functions, respectively. Namely, the identity will see the output as a linear combination of the previous layer nodes without further transformation (allowing it to take any value between negative and positive infinity, and hence making it an appropriate function for the type of outcome we seek to estimate).

Feedforward Neural Network - Forward Pass

Activation Funtion: Rectified Linear Unit (ReLU)

$$R(z) = max(0, z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases}$$

Weights:

$$w^1 = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}, \quad w^2 = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}, \quad w^3 = \begin{bmatrix} w_{1,1} \\ w_{2,1} \end{bmatrix}$$

Biases:

$$b^1 = \begin{bmatrix} b_1 & b_2 \end{bmatrix}, \quad b^2 = \begin{bmatrix} b_1 & b_2 \end{bmatrix}, \quad b^3 = b^3$$

Input Layer:

$$\vec{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

Hidden Layers:

$$z^1 = \begin{bmatrix} z_1 & z_2 \end{bmatrix}, \quad z^2 = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

Specifically[1],

$$z^1 = R(\vec{x} \cdot w^1 + b^1)$$

$$= R\Bigg(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} \Bigg)$$

$$= R\Bigg(x_1 \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \end{bmatrix} + x_2 \cdot \begin{bmatrix} w_{2,1} & w_{2,2} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} \Bigg) = R\Bigg(\begin{bmatrix} x_1 w_{1,1} + x_2 w_{2,1} + b_1 & x_1 w_{1,2} + x_2 w_{2,2} + b_2 \end{bmatrix} \Bigg)$$

$$= \Big[R(x_1 w_{1,1} + x_2 w_{2,1} + b_1) \quad R(x_1 w_{1,2} + x_2 w_{2,2} + b_2) \Big] = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

So,

$$z_1^1 = R(x_1 w_{1,1}^1 + x_2 w_{2,1}^1 + b_1^1)$$
$$z_2^1 = R(x_1 w_{1,2}^1 + x_2 w_{2,2}^1 + b_2^1)$$

Similarly,

$$z^2 = R(z^1 \cdot w^2 + b^2) = ... = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

$$z_1^2 = R(z_1^1 w_{1,1}^2 + z_2^1 w_{2,1}^2 + b_1^2)$$
$$z_2^2 = R(z_1^1 w_{1,2}^2 + z_2^1 w_{2,2}^2 + b_2^2)$$

Output Function: Identity

$$I(z) = I\big(b + \sum \alpha_i \cdot w_i\big) = z$$

Output Layer:

$$\hat{y} = I(z^3) = I\big(b^3 + \sum z^2 \cdot w^3\big)$$
$$\hat{y} = z^3 = z_1^2 w_{1,1}^3 + z_2^2 w_{2,1}^3 + b^3$$

Cost Funtion: MSE[2]

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Having randomized the first input layer and obtained the subsequent values in the feedforward network structure, we use back-propagation to minimize the cost function (MSE) via gradient descent and obtain a set of updated weights (that will reduce the mean error). Specifically, as the same suggests, we move through the network in a backward manner, starting with the total computed error, and taking recursive partial error derivatives (applying the chain rule) with respect to each of the weights, $w_{i,j}^k \in \{w^1, w^2, w^3\}$, to obtain the new $'w_{i,j}^k$, such that $'w_{i,j}^k = w_{i,j}^k - \partial J / \partial w_{i,j}^k$.

[1]Where weights, w, and biases, b, can be seen as the slope and intercept coefficients for the linear combination inputs of z.
[2]Where y is some target value for which $\hat{y}$ is the algorithm's prediction.

Derivatives:

1. ReLU Activation Function

$$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. MSE Cost Function[3]

$$J' = (\hat{y} - y)$$

Back-propagation

Starting with $w^3_{1,1}$, we compute its updated value, $'w^3_{1,1}$, as follows[4]:

$$\frac{\partial J}{\partial w^3_{1,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial w^3_{1,1}}$$

$$\frac{\partial J}{\partial z^3_{out}} = \frac{\partial}{\partial z^3_{out}} \left[\frac{1}{2}(y - \hat{y})^2\right]_{\hat{y} = z^3_{out}} = (y - z^3_{out}) \times \frac{\partial}{\partial z^3_{out}}[-z^3_{out}] = -(y - z^3_{out}) = z^3_{out} - y$$

$$\frac{\partial z^3_{in}}{\partial w^3_{1,1}} = \frac{\partial}{\partial w^3_{1,1}}\left[z^2_{1,out} w^3_{1,1} + z^2_{2,out} w^3_{2,1} + b^3\right] = z^2_{1,out}$$

$$\frac{\partial J}{\partial w^3_{1,1}} = (z^3_{out} - y) \times z^2_{1,out}$$

$$'w^3_{1,1} = w^3_{1,1} - \left[(z^3_{out} - y) \times z^2_{1,out}\right]$$

Similarly,

$$\frac{\partial J}{\partial w^3_{2,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial w^3_{2,1}} = (z^3_{out} - y) \times z^2_{2,out}$$

$$'w^3_{2,1} = w^3_{2,1} - \left[(z^3_{out} - y) \times z^2_{2,out}\right]$$

We move on to the second set of weights, starting with $w^2_{1,1}$, to obtain $'w^2_{1,1}$:

$$\frac{\partial J}{\partial w^2_{1,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{1,out}} \times \frac{\partial z^2_{1,out}}{\partial z^2_{1,in}} \times \frac{\partial z^2_{1,in}}{\partial w^2_{1,1}}$$

$$\frac{\partial J}{\partial z^3_{out}} = z^3_{out} - y$$

$$\frac{\partial z^3_{in}}{\partial z^2_{1,out}} = \frac{\partial}{\partial z^2_{1,out}}\left[z^2_{1,out} w^3_{1,1} + z^2_{2,out} w^3_{2,1} + b^3\right] = w^3_{1,1}$$

[3] Assuming $n = 1$ such that $J = \frac{1}{2}(y - \hat{y})^2$, where $\frac{1}{2}$ is a constant added for simpler derivative calculations.
[4] Note that $z^3_{in} = z^3_{out}$, so we need not include $\partial z^3_{out} / \partial z^3_{in} = 1$.

$$\frac{\partial z^2_{1,out}}{\partial z^2_{1,in}} = \frac{\partial}{\partial z^2_{1,in}}\left[R(z^2_{1,in})\right] = \begin{cases} 1 & z^2_{1,in} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial z^2_{1,in}}{\partial w^2_{1,1}} = \frac{\partial}{\partial w^2_{1,1}}\left[z^1_{1,out}w^2_{1,1} + z^1_{2,out}w^2_{2,1} + b^2_1\right] = z^1_{1,out}$$

$$\frac{\partial J}{\partial w^2_{1,1}} = (z^3_{out} - y) \times w^3_{1,1} \times (1,0) \times z^1_{1,out} = \left((z^3_{out} - y) \times w^3_{1,1} \times z^1_{1,out}, 0\right)$$

$$'w^2_{1,1} = w^2_{1,1} - \left((z^3_{out} - y) \times w^3_{1,1} \times z^1_{1,out}, 0\right)$$

Similarly,

$$\frac{\partial J}{\partial w^2_{2,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{1,out}} \times \frac{\partial z^2_{1,out}}{\partial z^2_{1,in}} \times \frac{\partial z^2_{1,in}}{\partial w^2_{2,1}} = \left((z^3_{out} - y) \times w^3_{1,1} \times z^1_{2,out}, 0\right)$$

$$\frac{\partial J}{\partial w^2_{1,2}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{2,out}} \times \frac{\partial z^2_{2,out}}{\partial z^2_{2,in}} \times \frac{\partial z^2_{2,in}}{\partial w^2_{1,2}} = \left((z^3_{out} - y) \times w^3_{2,1} \times z^1_{1,out}, 0\right)$$

$$\frac{\partial J}{\partial w^2_{2,2}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{2,out}} \times \frac{\partial z^2_{2,out}}{\partial z^2_{2,in}} \times \frac{\partial z^2_{2,in}}{\partial w^2_{2,2}} = \left((z^3_{out} - y) \times w^3_{2,1} \times z^1_{2,out}, 0\right)$$

$$'w^2_{2,1} = w^2_{2,1} - \left((z^3_{out} - y) \times w^3_{1,1} \times z^1_{2,out}, 0\right)$$
$$'w^2_{1,2} = w^2_{1,2} - \left((z^3_{out} - y) \times w^3_{2,1} \times z^1_{1,out}, 0\right)$$
$$'w^2_{2,2} = w^2_{2,2} - \left((z^3_{out} - y) \times w^3_{2,1} \times z^1_{2,out}, 0\right)$$

The last set of weights arguably poses the greatest complexity in computing partial derivatives as they influence the error/cost function implicitly through both neurons in the second hidden layer. Although this forces us to compute additional partial derivatives, we've luckily already computed a great portion of them in the previous steps. That said, we start with $w^1_{1,1}$, to obtain $'w^1_{1,1}$, as follows:

$$\frac{\partial J}{\partial w^1_{1,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{out}} \times \frac{\partial z^2_{out}}{\partial z^2_{in}} \times \frac{\partial z^2_{in}}{\partial z^1_{1,out}} \times \frac{\partial z^1_{1,out}}{\partial z^1_{1,in}} \times \frac{\partial z^1_{1,in}}{\partial w^1_{1,1}}$$

where,

$$\frac{\partial z^3_{in}}{\partial z^2_{out}} = \frac{\partial z^3_{in}}{\partial z^2_{1,out}} + \frac{\partial z^3_{in}}{\partial z^2_{2,out}}$$

$$\frac{\partial z^2_{out}}{\partial z^2_{in}} = \frac{\partial z^2_{1,out}}{\partial z^2_{1,in}} + \frac{\partial z^2_{2,out}}{\partial z^2_{2,in}}$$

$$\frac{\partial z^2_{in}}{\partial z^1_{1,out}} = \frac{\partial z^2_{1,in}}{\partial z^1_{1,out}} + \frac{\partial z^2_{2,in}}{\partial z^1_{1,out}}$$

$$\frac{\partial J}{\partial z^3_{out}} = z^3_{out} - y$$

$$\frac{\partial z^3_{in}}{\partial z^2_{out}} = \frac{\partial}{\partial z^2_{1,out}}\left[z^2_{1,out}w^3_{1,1} + z^2_{2,out}w^3_{2,1} + b^3\right] + \frac{\partial}{\partial z^2_{2,out}}\left[z^2_{1,out}w^3_{1,1} + z^2_{2,out}w^3_{2,1} + b^3\right] = w^3_{1,1} + w^3_{2,1}$$

$$\frac{\partial z^2_{out}}{\partial z^2_{in}} = \begin{cases} 1 & z^1_{1,out}w^2_{1,1} + z^1_{2,out}w^2_{2,1} + b^2_1 > 0 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} 1 & z^1_{1,out}w^2_{1,2} + z^1_{2,out}w^2_{2,2} + b^2_2 > 0 \\ 0 & \text{otherwise} \end{cases} = (2,1,0)$$

$$\frac{\partial z^2_{in}}{\partial z^1_{1,out}} = \frac{\partial}{\partial z^1_{1,out}}\left[z^1_{1,out}w^2_{1,1} + z^1_{2,out}w^2_{2,1} + b^2_1\right] + \frac{\partial}{\partial z^1_{1,out}}\left[z^1_{1,out}w^2_{2,1} + z^1_{2,out}w^2_{2,2} + b^2_2\right] = w^2_{1,1} + w^2_{2,1}$$

$$\frac{\partial z^1_{1,out}}{\partial z^1_{1,in}} = \begin{cases} 1 & x_1 w^1_{1,1} + x_2 w^1_{2,1} + b^1_1 > 0 \\ 0 & \text{otherwise} \end{cases} = (1,0)$$

$$\frac{\partial z^1_{1,in}}{\partial w^1_{1,1}} = \frac{\partial}{\partial w^1_{1,1}}\left[x_1 w^1_{1,1} + x_2 w^1_{2,1} + b^1_1\right] = x_1$$

$$\frac{\partial J}{\partial w^1_{1,1}} = (z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (2,1,0) \times (w^2_{1,1} + w^2_{2,1}) \times (1,0) \times x_1$$

$$= \Big(2\big[(z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_1\big], \big[(z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_1\big], 0\Big)$$

$$= \big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_1\big) \times (2,1,0)$$

$$\text{let } A = (z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_1$$

$$\frac{\partial J}{\partial w^1_{1,1}} = \begin{cases} 2A & z^2_{in}, z^1_{1,in} > 0 \\ A & z^2_{1,in}, z^1_{1,in} > 0 \\ A & z^2_{2,in}, z^1_{1,in} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$'w^1_{1,1} = w^1_{1,1} - \big(2A, A, 0\big)$$

Similarly,

$$\frac{\partial J}{\partial w^1_{2,1}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{out}} \times \frac{\partial z^2_{out}}{\partial z^2_{in}} \times \frac{\partial z^2_{in}}{\partial z^1_{1,out}} \times \frac{\partial z^1_{1,out}}{\partial z^1_{1,in}} \times \frac{\partial z^1_{1,in}}{\partial w^1_{2,1}} = \big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_2\big) \times (2,1,0)$$

$$\frac{\partial J}{\partial w^1_{1,2}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{out}} \times \frac{\partial z^2_{out}}{\partial z^2_{in}} \times \frac{\partial z^2_{in}}{\partial z^1_{2,out}} \times \frac{\partial z^1_{2,out}}{\partial z^1_{2,in}} \times \frac{\partial z^1_{2,in}}{\partial w^1_{1,2}} = \big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{2,1} + w^2_{2,2}) \times x_1\big) \times (2,1,0)$$

$$\frac{\partial J}{\partial w^1_{2,2}} = \frac{\partial J}{\partial z^3_{out}} \times \frac{\partial z^3_{in}}{\partial z^2_{out}} \times \frac{\partial z^2_{out}}{\partial z^2_{in}} \times \frac{\partial z^2_{in}}{\partial z^1_{2,out}} \times \frac{\partial z^1_{2,out}}{\partial z^1_{2,in}} \times \frac{\partial z^1_{2,in}}{\partial w^1_{2,2}} = \big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{2,1} + w^2_{2,2}) \times x_2\big) \times (2,1,0)$$

$$'w^1_{2,1} = w^1_{2,1} - \Big[\big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{1,1} + w^2_{2,1}) \times x_2\big) \times (2,1,0)\Big]$$

$$'w^1_{1,2} = w^1_{1,2} - \Big[\big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{2,1} + w^2_{2,2}) \times x_1\big) \times (2,1,0)\Big]$$

$$'w^1_{2,2} = w^1_{2,2} - \Big[\big((z^3_{out} - y) \times (w^3_{1,1} + w^3_{2,1}) \times (w^2_{2,1} + w^2_{2,2}) \times x_2\big) \times (2,1,0)\Big]$$

Problem 2: Random Forests and Cross-Validation

Using the "**diabetes**" data, create a 75/25 train and test split of the data. Then, fit a random forest with `mtry` $\in \{p, p/2\}$ and `max.depth` $\in \{4, 6, 8, 10\}$ for all combinations of `mtry` and `max.depth` and with the other parameters as set in the example below. Compare the performance and describe which setting you would use and what your results indicate.

```
rf_mod <- ranger(early_readmit~., data=train, mtry=p, max.depth=6,
                 importance="impurity", num.trees=100,
                 min.node.size=50, class.weights=c(1.10, 11))
```

Then, take a balanced random subset of 1,000 points in each class and fit a random forest with the same parameters but without class weights. Plot the OOB error from 1 to 200 trees and explain why you see the trend that you do. You may use the OOB Curve to do this or write a short function to do so.

Solution

Based on results from Assignment 4, we use the following 13 variables for predicting early readmission:

```
early_readmit ~ encounter_type + race_group + age_group + discharge_to_home +
                admission_source + time_in_hospital + med_specialty +
                num_lab_procedures + number_emergency + number_inpatient +
                number_diagnoses + diabetesMed + diag_1_cat
```

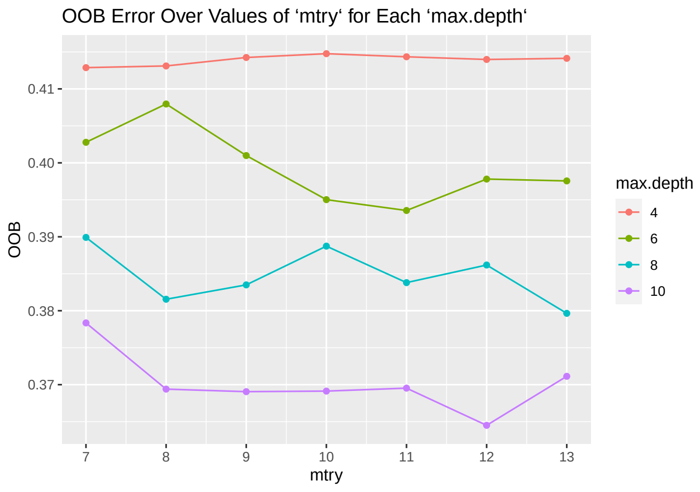
Thus, given that $p = 13$, we proceed by fitting a series of random forest models corresponding to each combination of `mtry` $\in \{7, 8, 9, 10, 11, 12, 13\}$ and `max.depth` $\in \{4, 6, 8, 10\}$ in tandem with the remaining parameters specified above. To find the best fitting model for our purposes, we compare each model's performance in terms of to OOB error, sensitivity, specificity, accuracy, and AUC score. The results based on these metrics are given by Table 1 below. Moreover, Table 2 provides the `mtry` and `max.depth` combinations for which model performance was best in each category. For visualization purposes, we also graph the changes in OOB error, accuracy, and AUC with respect to possible values of `mtry` for each `max.depth` value.

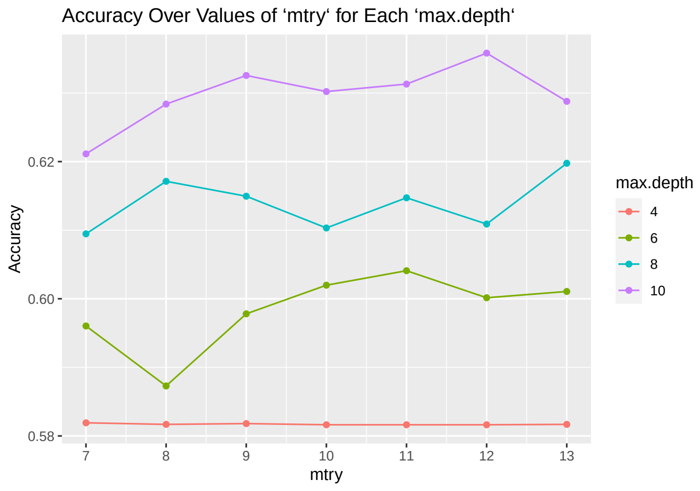
Table 1: Model Performance for Values of 'mtry' and 'max.depth'

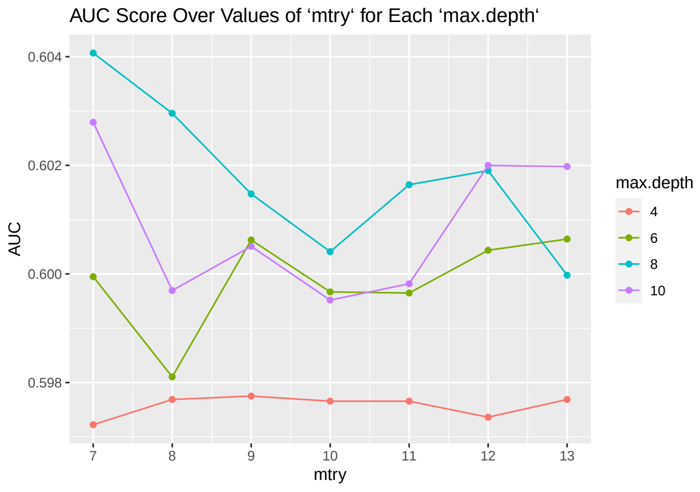
mtry	max.depth	OOB	MIV	FN	TP	Sensitivity	Specificity	Accuracy	AUC
7	4	0.4129	number_inpatient	6725	944	0.6158	0.5787	0.5819	0.5972
7	6	0.4028	number_inpatient	6461	927	0.6047	0.5952	0.5960	0.5999
7	8	0.3899	number_inpatient	6215	916	0.5975	0.6106	0.6095	0.6041
7	10	0.3784	num_lab_procedures	5985	890	0.5806	0.6250	0.6211	0.6028
8	4	0.4131	number_inpatient	6731	946	0.6171	0.5783	0.5817	0.5977
8	6	0.4080	number_inpatient	6624	937	0.6112	0.5850	0.5873	0.5981
8	8	0.3816	number_inpatient	6063	898	0.5858	0.6201	0.6171	0.6030
8	10	0.3694	num_lab_procedures	5834	866	0.5649	0.6345	0.6284	0.5997
9	4	0.4142	discharge_to_home	6729	946	0.6171	0.5784	0.5818	0.5978
9	6	0.4010	number_inpatient	6429	926	0.6040	0.5972	0.5978	0.6006
9	8	0.3835	number_inpatient	6100	897	0.5851	0.6178	0.6150	0.6015
9	10	0.3691	num_lab_procedures	5756	861	0.5616	0.6394	0.6326	0.6005
10	4	0.4148	discharge_to_home	6732	946	0.6171	0.5782	0.5816	0.5977
10	6	0.3950	number_inpatient	6345	915	0.5969	0.6025	0.6020	0.5997
10	8	0.3887	number_inpatient	6186	902	0.5884	0.6124	0.6103	0.6004
10	10	0.3691	num_lab_procedures	5798	862	0.5623	0.6367	0.6302	0.5995
11	4	0.4143	number_inpatient	6732	946	0.6171	0.5782	0.5816	0.5977
11	6	0.3936	number_inpatient	6304	911	0.5943	0.6050	0.6041	0.5996
11	8	0.3838	number_inpatient	6105	898	0.5858	0.6175	0.6147	0.6016
11	10	0.3695	num_lab_procedures	5778	861	0.5616	0.6380	0.6313	0.5998
12	4	0.4140	number_inpatient	6731	945	0.6164	0.5783	0.5816	0.5974
12	6	0.3978	number_inpatient	6383	921	0.6008	0.6001	0.6001	0.6004
12	8	0.3862	number_inpatient	6180	906	0.5910	0.6128	0.6109	0.6019
12	10	0.3645	num_lab_procedures	5698	860	0.5610	0.6430	0.6358	0.6020
13	4	0.4141	number_inpatient	6731	946	0.6171	0.5783	0.5817	0.5977
13	6	0.3976	number_inpatient	6366	920	0.6001	0.6012	0.6011	0.6006
13	8	0.3796	number_inpatient	6002	883	0.5760	0.6240	0.6198	0.6000
13	10	0.3711	num_lab_procedures	5834	873	0.5695	0.6345	0.6288	0.6020

Table 2: Best Combination for Each Metric

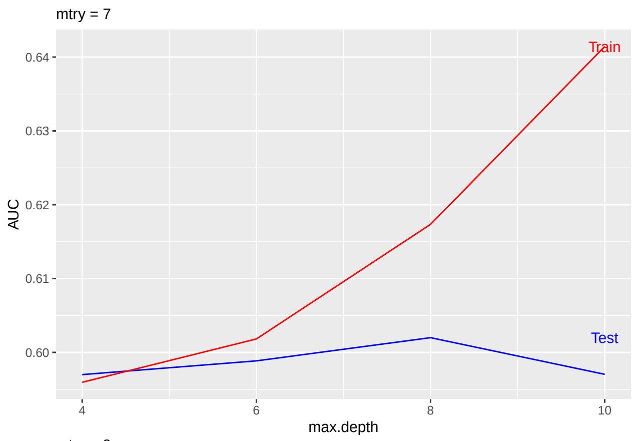
mtry	max.depth	Metric
12	10	OOB Error
8-11, 13	4	Sensitivity
12	10	Specificity
12	10	Accuracy
7	8	AUC

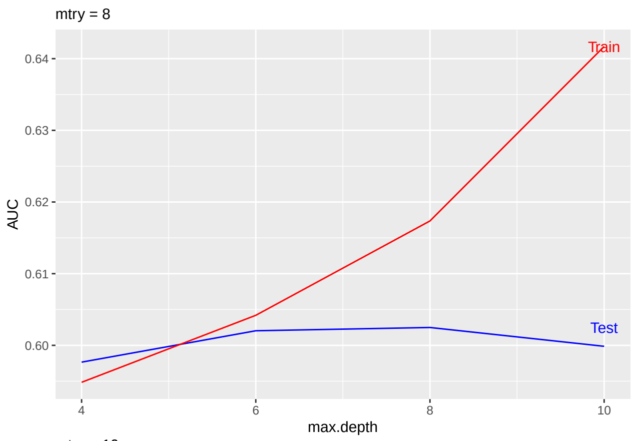


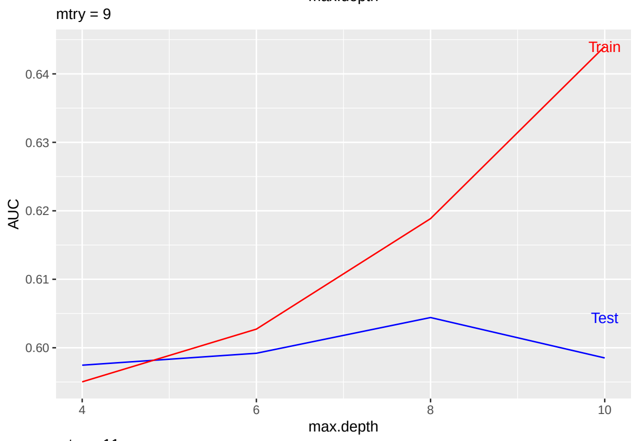


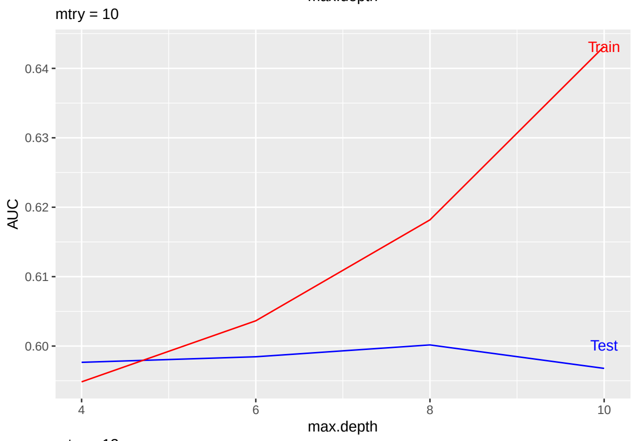


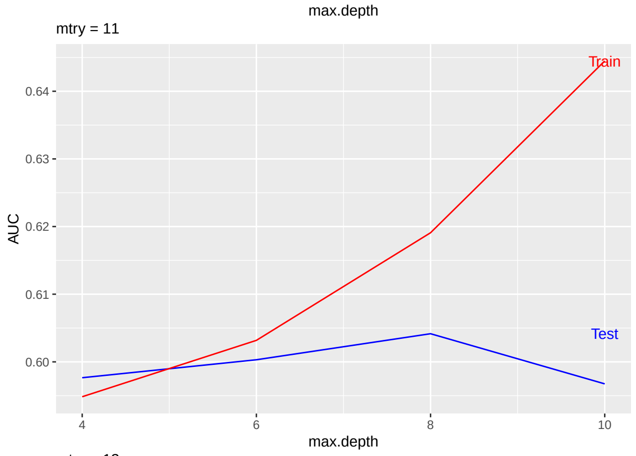
Given these results, it is evident that `max.depth`=4 results in the lowest model performance, while `max.depth` values of 8 and 10 produce better but fluctuating outcomes over values of `mtry`. However, it is possible for the high model performance corresponding to `max.depth`=10 to be a result of overfitting. To investigate this further, we record and plot AUC scores for both train and test sets to compare their changes over values of `max.depth` for each value of `mtry`.

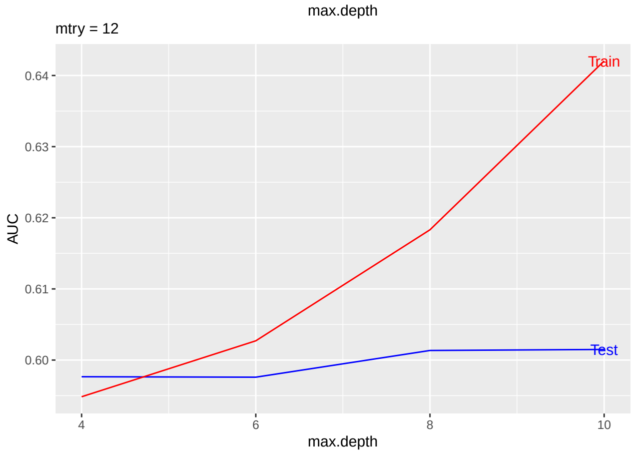

mtry = 7
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test

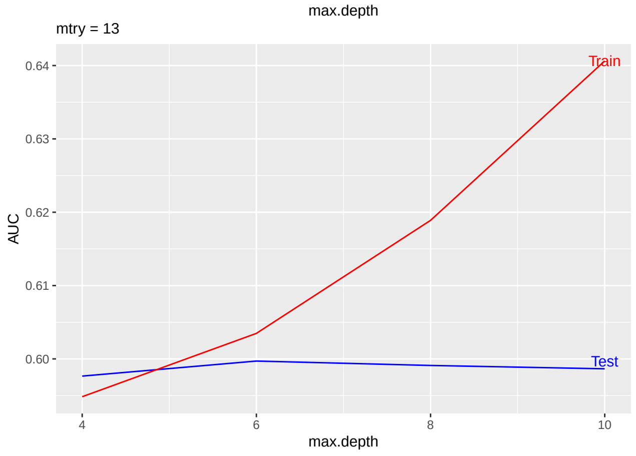

mtry = 8
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test


mtry = 9
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test


mtry = 10
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test


mtry = 11
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test


mtry = 12
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test

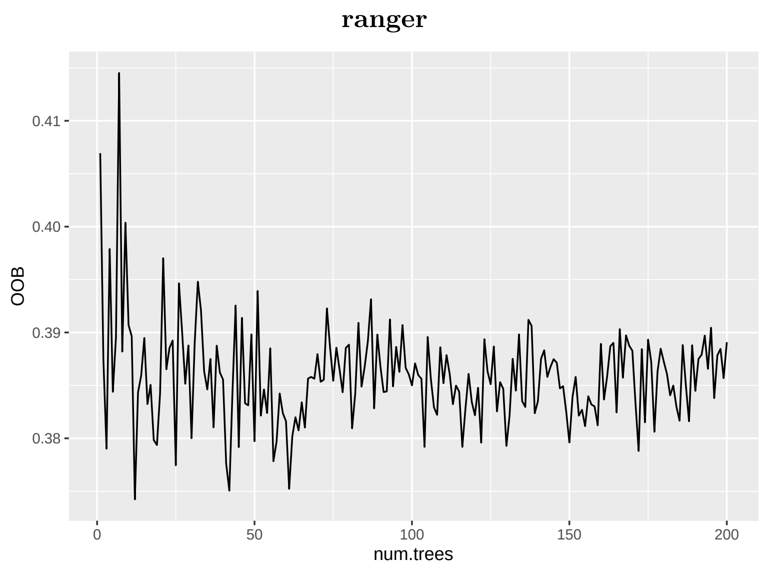

mtry = 13
0.64
0.63
0.62
0.61
0.60
AUC
4
6
8
10
max.depth
Train
Test

Given the discrepancy in AUC scores for larger values of `max.depth`, it is evident that the model begins to overfit the data after splitting results in maximum tree depths larger than 6. However, since these values peak at `max.depth`=8 for the test set we find it most fitting for our random forest model. To find a promising corresponding `mtry` value, we direct our attention to the following subset of results, which show that `mtry`=8 is the best compromise to minimize the possibility of over or underfitting.

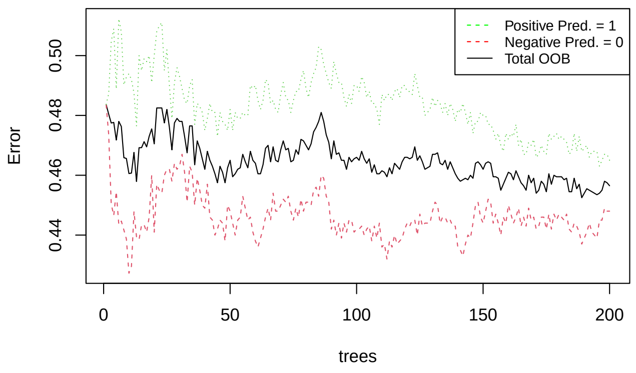
Table 3: Model Performance for Each 'mtry' and 'max.depth'=8

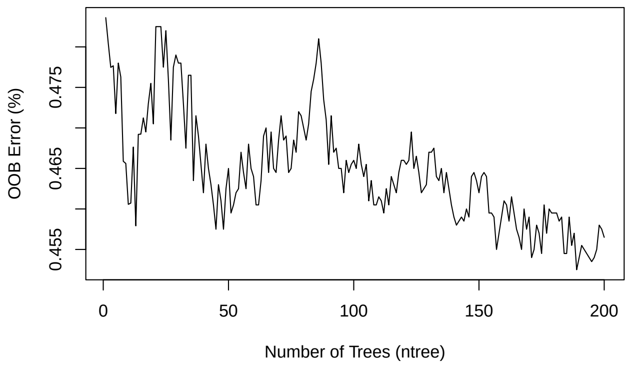
mtry	OOB	Sensitivity	Specificity	Accuracy	AUC
7	0.3899	0.5975	0.6106	0.6095	0.6041
8	0.3816	0.5858	0.6201	0.6171	0.6030
9	0.3835	0.5851	0.6178	0.6150	0.6015
10	0.3887	0.5884	0.6124	0.6103	0.6004
11	0.3838	0.5858	0.6175	0.6147	0.6016
12	0.3862	0.5910	0.6128	0.6109	0.6019
13	0.3796	0.5760	0.6240	0.6198	0.6000

With our chosen values for `max.depth` and `mtry` (both set to 8), we proceed by generating a random subset of our train data with 1,000 observations from each class (positive and negative) with which to train our model, excluding class weights, as a different approach to account for imbalance in our data. With this new model, we subsequently appeal to OOB error as a way to find an appropriate number of trees to use in the random forest. Specifically, we graph the OOB error obtained from setting `num.trees` equal to each value in 1-200. Since the `ranger` package, unlike `randomForest`, does not provide a vector of prediction errors calculated after each additional tree is included in the model, the final OOB error is calculated for a series of random forest models with the corresponding number of trees. Moreover, despite not having a `max_depth` parameter, we use `randomForest` with all other parameters (and train data) the same, to obtain this graph for comparative purposes.



randForest





Here we see that when `ranger` was used, OOB error fluctuated between 0.38 and 0.4, while fluctuating between 0.46 and 0.48 when `randomForest` was used, across values of `num.trees`. In either case, it is difficult to determine a point at which OOB error begins to stabilize, despite this window becoming narrower for increasing values of `num.trees`. This indicates that, given our specified parameters and the data used to train this model, we will not see a significant decrease in prediction error when more trees are included. And, since changing this parameter drastically will yield minimal improvement in model performance, it follows that choosing a smaller number of trees will, at least, reduce computational cost. For this reason, it seems that a value for `num.trees` near 100 may be an appropriate choice considering the patterns noticed above.

Having tuned all of our random forest model parameters, we proceed by comparing both approaches to handling imbalance in the data. That is, (with `ranger`) using class weights akin to their inverse proportions in the train data, as well as using a balanced subset of the train data (of size 2,000). The corresponding values, obtained form using the metrics for judging model performance previously, are given by Table 3 below.

Table 4: Model Performance for Each Approach to Data Imbalance

Approach	OOB	MIV	FN	TP	Sensitivity	Specificity	Accuracy	AUC
Class Weights	0.3881	number_inpatient	6196	905	0.5903	0.6118	0.6099	0.6011
Balanced Train Set	0.4275	num_lab_procedures	6031	855	0.5577	0.6221	0.6165	0.5899

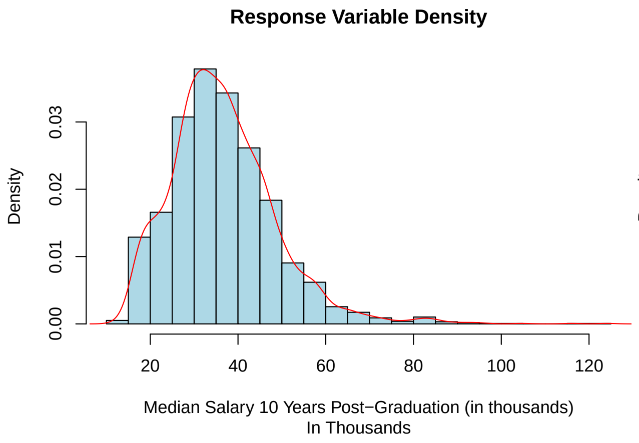
Note:
'mtry'='max.depth'=8 and 'num.trees'=100

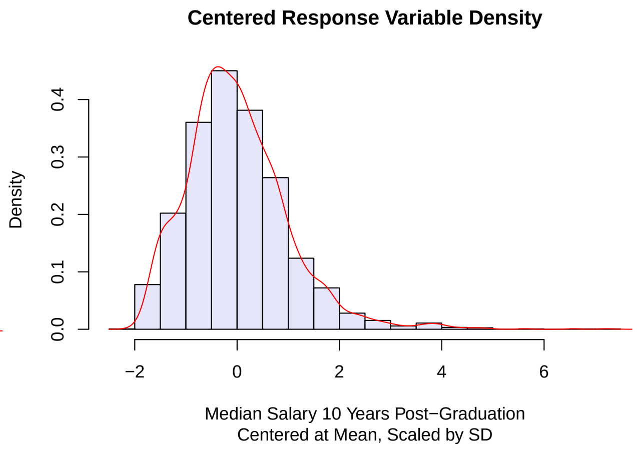
From these results, it is evident that, despite the similarities, the weights approach resulted in overall better model performance. This could, in part, be due to the specific subset of the train data used for training the random forest model in the balanced subset approach. But, more generally, it is reasonable to infer that this discrepancy in values results from a loss of information when sub-setting the train data. That is, while the model was exposed to all the information in the train data in the weights approach, it was only able to make predictions based on roughly 3.8% of this data in the latter approach, hence, making it less accurate and vastly more prone to error. Still however, it is striking to see such a closeness in model performance, given the small amount of data used by the model in the balanced train set approach. More than anything, this speaks to the overall nature of the data itself in regards to the outcome of interest. Namely, the limited amount of information that can be obtained from it and the predictive power its variables have over the response.

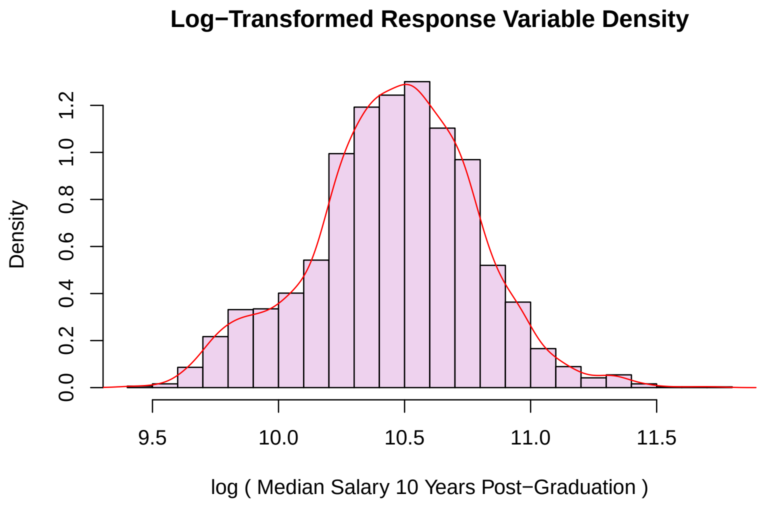
Problem 3: Deep Learning

The file `college_scorecard.csv` in the Data folder on Canvas contains data from the 2021 College Scorecard data from the Department of Education. This data is collected to help students evaluate potential colleges and identify predatory colleges that offer little benefit to enrolled students. Data variable descriptions are given in college data `dictionary.csv`. You can read more about the data here. First, plot and comment on the distribution for the outcome of interest, the median salary of a federally-funded student 10 years after graduation. Then, build a feedforward neural network to predict this outcome and another network to predict the log median salary. Compare the performance and architecture/structure of these models. You do not have to do any exploratory analysis of the data, but you should consider how you will explore different options for your network in a systematic way. Last, use the same architecture but remove variables related to student's own financial background. Comment on how the performance changes - do these seem to be important predictors?

Solution







Looking at the density plots of the response and log-transformed response variables above, we see that the former is heavily skewed to the right, likely following an exponential distribution. On the other hand, we notice that the latter is more evenly distributed about it's mean and more closely resembles a normal distribution. Given that this variable pertains to median income, these patterns in density further suggest that our response is log-normally distributed.

Feed-forward Neural Network 1: Response

Model Outline:

- Model Architecture:
  - Input Layer: 64 nodes
  - Hidden Layer 1: 64 nodes, ReLu activation function
  - Hidden Layer 2: 64 nodes, ReLu activation function
  - Hidden Layer 3: 32 nodes, ReLu activation function
  - Hidden Layer 4: 32 nodes, ReLu activation function
  - Output Layer: 1 node, identity output/activation function
- Back-propagation:
  - Optimizer: RMSprop
  - Loss Function: Mean Squared Error (MSE)
  - Performance Metric: Mean Absolute Error (MAE)
- Training:
  - Ephocs: 40
  - Batch Size: 100 (Mini-Batch SGD)
  - CV Holdout Set: 0.2

Code:

```
cs_dnn <- keras_model_sequential() %>%

  # model architecture
  layer_dense(units=ncol(cs_train_x),
              activation="relu",
              input_shape=ncol(cs_train_x)) %>%
  layer_dense(units=ncol(cs_train_x),
              activation="relu") %>%
  layer_dense(units=ncol(cs_train_x)/2,
              activation="relu") %>%
  layer_dense(units=ncol(cs_train_x)/2,
              activation="relu") %>%
  layer_dense(units=1) %>%

  # back-propagation
  compile(optimizer="rmsprop",
          loss="mse",
          metrics=c("mae")) %>%

  # training
  fit(x=cs_train_x,
      y=cs_train_y,
      epochs=40,
      batch_size=100,
      validation_split=0.2,
      callbacks=callback_early_stopping(patience=4))
```

Model Performance and Predictions:

Loss/MAE Curves Before and After Tuning Model Parameters

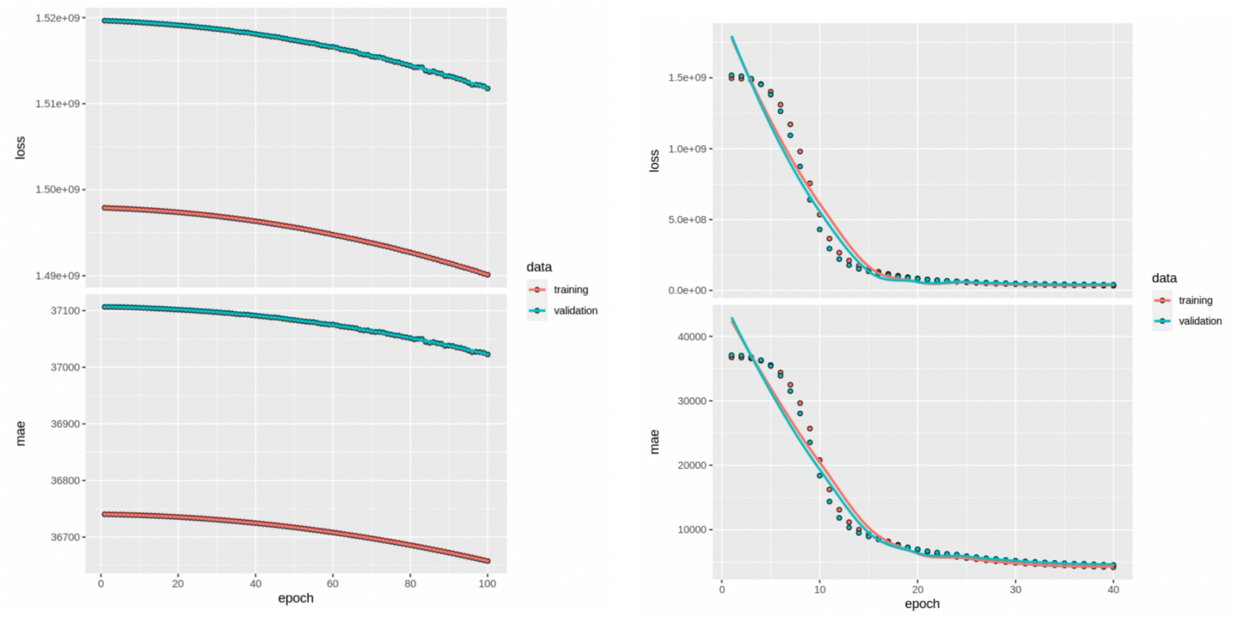


Table 5: Model Performance Before and After Tuning Parameters

	Loss	MAE
Before	1426709504	35837.312
After	41050780	4370.487

Table 6: First 10 Obeserved vs.Predicted Responses

Observed	Predicted	% Error
33300	34234.37	2.8059159
28900	31934.74	10.5008304
26100	26417.44	1.2162452
26500	26235.96	-0.9963774
34200	39618.32	15.8430409
27900	25096.07	-10.0499283
27800	29677.95	6.7552158
35500	40391.52	13.7789296
39800	30069.89	-24.4475126
23900	26403.80	10.4761506

Feed-forward Neural Network 2: Log-Transformed Response

Outline:

- Model Architecture:
  - Input Layer: 64 nodes
  - Hidden Layer 1: 64 nodes, ReLu activation function, layer dropout rate of 0.2, batch normalization
  - Hidden Layer 2: 16 nodes, ReLu activation function, layer dropout rate of 0.2, batch normalization
  - Hidden Layer 3: 8 nodes, ReLu activation function, layer dropout rate of 0.2, batch normalization
  - Output Layer: 1 node, identity output/activation function
- Back-propagation:
  - Optimizer: RMSprop
  - Loss Function: Mean Squared Error (MSE)
  - Performance Metric: Mean Absolute Error (MAE)
- Training:
  - Ephocs: 50
  - Batch Size: 50 (Mini-Batch SGD)
  - CV Holdout Set: 0.2

Code:

```
cs_dnn2 <- keras_model_sequential() %>%

  # model architecture
  layer_dense(units=ncol(cs_train2_x),
              activation="relu",
              input_shape=ncol(cs_train2_x)) %>%
  layer_dropout(rate=0.2) %>%
  layer_batch_normalization() %>%
  layer_dense(units=16,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>%
  layer_batch_normalization() %>%
  layer_dense(units=8,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>%
  layer_batch_normalization() %>%
  layer_dense(units=1) %>%

  # back-propagation
  compile(optimizer="rmsprop",
          loss="mse",
          metrics=c("mae")) %>%

  # training
  fit(x=cs_train2_x,
      y=cs_train2_y,
      epochs=50,
      batch_size=50,
      validation_split=0.2,
      callbacks=callback_early_stopping(patience=4))
```

Model Performance and Predictions:

Loss/MAE Curves Before and After Tuning Model Parameters

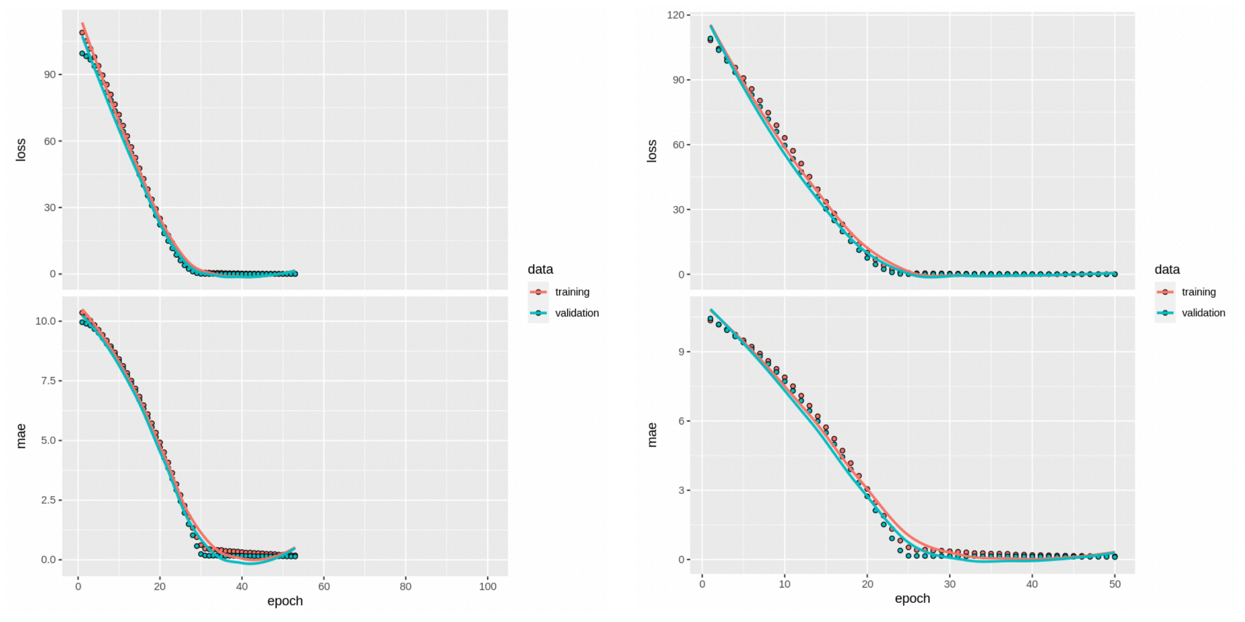


Table 7: Model Performance Before and After Tuning Parameters

	Loss	MAE
Before	0.0355498	0.1462143
After	0.0197238	0.1007187

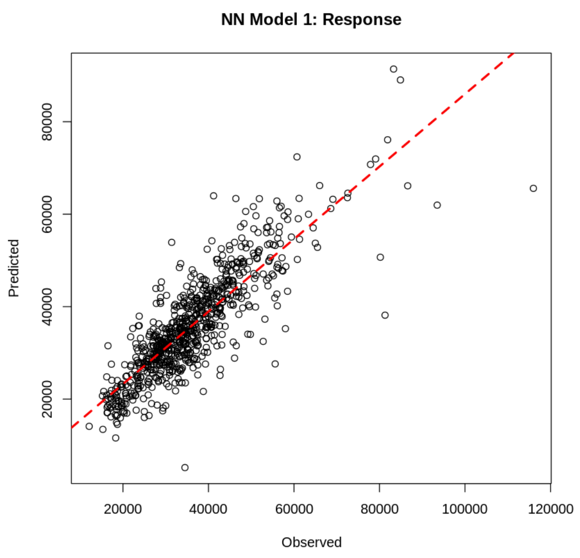
Table 8: First 10 Obeserved vs.Predicted Responses

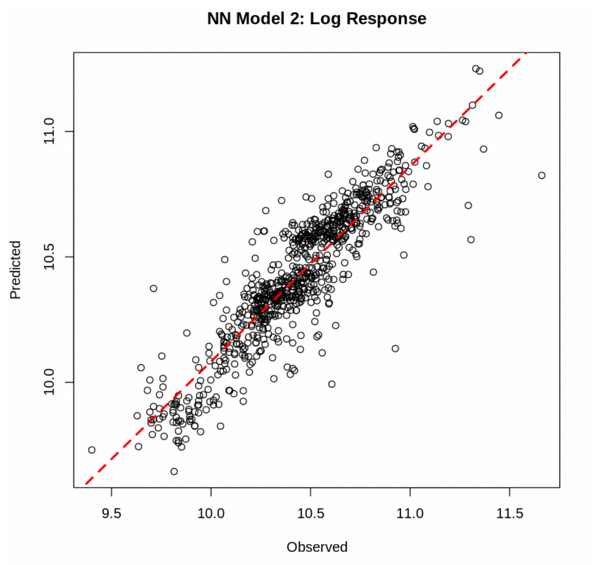
Observed	Predicted	% Error
10.41331	10.49596	0.7936987
10.27160	10.28729	0.1527915
10.16969	10.23869	0.6784612
10.18490	10.18832	0.0335986
10.43998	10.52845	0.8474257
10.23638	10.28275	0.4530217
10.23279	10.24520	0.1212836
10.47729	10.75023	2.6050351
10.59162	10.45953	-1.2471384
10.08163	10.18010	0.9766995

Feed-forward Neural Network Model Comparisons

Table 9: NN Model Performance Comparison

	MAE	MA%E	RSE	R2
Model 1: Response	4370.4873	12.7544	5943.0000	0.7154
Model 2: Log Response	0.1007	0.9607	0.1218	0.8086





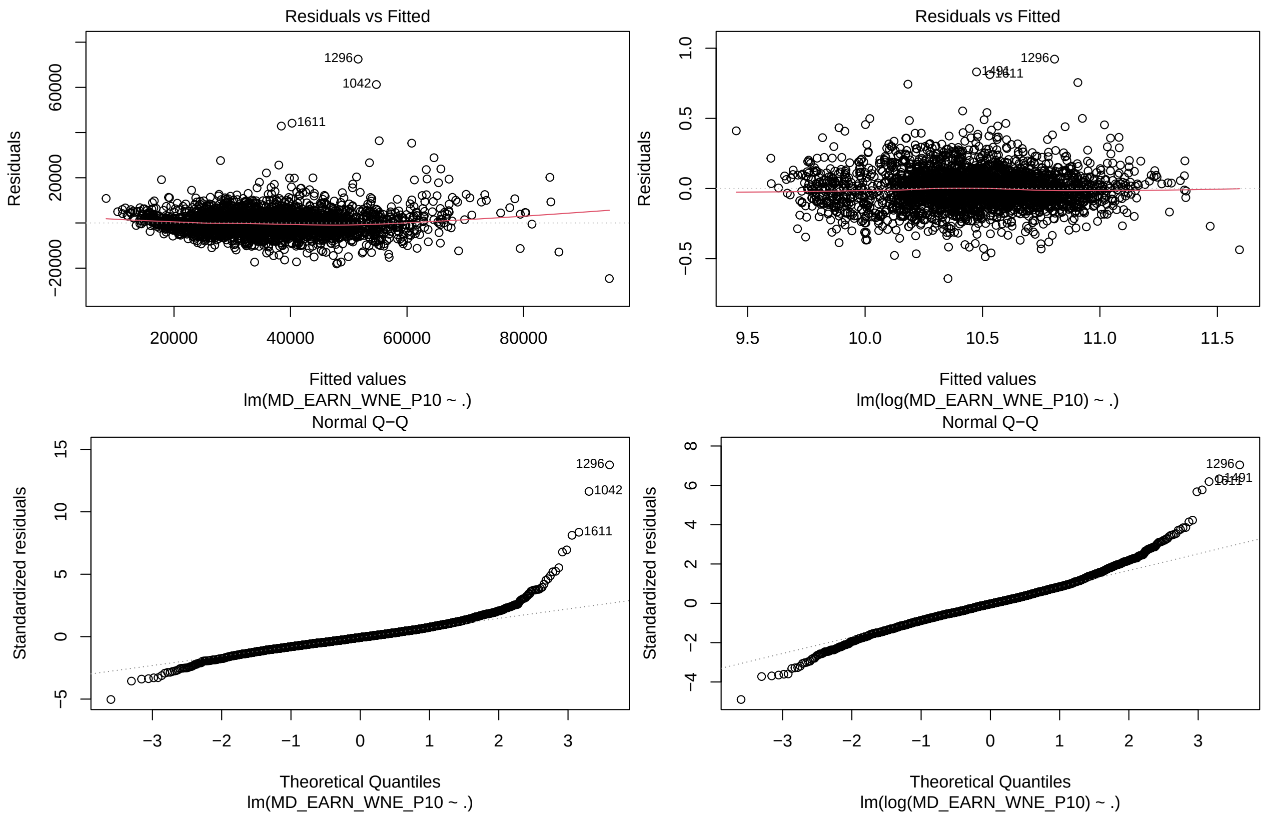
Assessing the overall performance of both neural networks, we see that even after tuning model parameters, the second model performed significantly better. Not only did it produce a higher R^2 value, indicating superior goodness-of-fit (and hence, a stronger relationship between predictors and the target variable), but it had a mean absolute percentage error[5] (in test set predictions) of roughly 1%, while the first model showed a mean absolute percentage error of nearly 13%.

The reason for this discrepancy in model performance can be attributed to the log-scale transformation of our target variable (which we noted previously likely follows a log-normal distribution). Specifically, since our unchanged response has an approximately exponential distribution, it does not reflect the kind of linearity assumed by traditional linear regression, and hence, produces residuals that are not normally distributed.

Contrastingly, seeing the response on a logarithmic scale, preserves the linear relationship between dependent and independent variables, allowing the neural network to obtain more accurate results. Despite not requiring that this assumption be met to perform effectively, given the nature of computation in neural networks used for predicting continuous outcomes (regression problems, where MSE is the objective/loss function), it follows that performance may improve when there is smaller residual deviance.

[5]Note that this metric was used to relativize mean absolute error between models (which measured the response variable on different scales).

Looking at the residuals vs. fitted plots as well as the Normal Q-Q plots for simple linear regression on our target variable (left) and its log-transformation (right) below, this contrast in regression assumptions of linearity and residual normality is made more clear.



Each of the models' architectures and optimal parameters can also be attributed to this difference in response scales. For example, we notice that including layer dropouts to prevent overfitting due to noise in the first model further decreases its performance. This is likely due to the fact that what the model would consider "noise" may in fact be the set of comparatively large response values that are crucial for making accurate predictions.

Similarly, it is likely that this model demonstrated better performance when increasing the number of hidden layers and nodes in each layer (and hence, the number of parameters), as it ultimately had to fit a more complex (non-linear) function than the second model. Thus, although not necessary for producing a working model, meeting these crucial regression assumptions, can in fact significantly improve a neural network's performance in similar scenarios.

Feed-forward Neural Network 2b: Log-Transformed Response

Removing Individual Financial Background Predictors

To evaluate the importance of factors related to a student's own financial background in predicting median earnings 10 years after graduation, we remove the following predictors from the data set:

- Share of students who received a Pell Grant while in school (`PELL_EVER`)
- Median household income (`MEDIAN_HH_INC`)
- Poverty rate, via Census data (`POVERTY_RATE`)
- Unemployment rate, via Census data (`UNEMP_RATE`)

Moreover, we use this subset of the data to train a feed-forward neural network of the same structure as the second model[6], using the log-transformed response, for comparative purposes in evaluating this effect. The results for both models are given below.

Model Performance and Predictions:

Loss/MAE Curves Before and After Removing Predictors

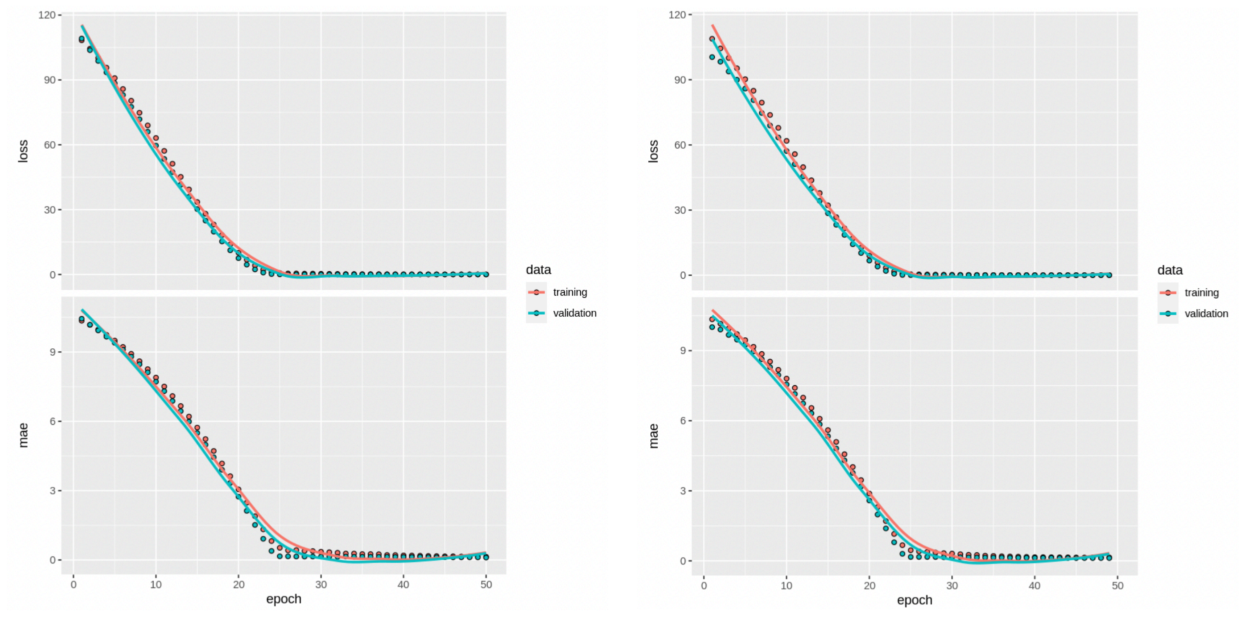


Table 10: Model Performance Before and After Removing Predictors

	Loss	MAE	MA%E	RSE	R2
Model 2	0.0197	0.1007	0.9607	0.1218	0.8086
Model 2b	0.0230	0.1103	1.0568	0.1137	0.7912

[6]Note that this model has 60 rather than 64 nodes in the input and first hidden layers due to removing individual financial background predictors.

Table 11: Comparing First 10 Obeserved vs.Predicted Responses

Observed	Predicted 2	Predicted 2b	% Error 2	% Error 2b
10.41331	10.49596	10.51609	0.7936987	0.9869417
10.27160	10.28729	10.29693	0.1527915	0.2466133
10.16969	10.23869	10.37494	0.6784612	2.0182857
10.18490	10.18832	10.19042	0.0335986	0.0542469
10.43998	10.52845	10.46593	0.8474257	0.2486027
10.23638	10.28275	10.30050	0.4530217	0.6263544
10.23279	10.24520	10.24692	0.1212836	0.1380923
10.47729	10.75023	10.68917	2.6050351	2.0222984
10.59162	10.45953	10.24457	-1.2471384	-3.2766481
10.08163	10.18010	10.30204	0.9766995	2.1861661

Given these results, it is clear that model performance only decreased slightly after removing individual financial background predictors. Specifically, the R^2 values shown in Table 10 indicate that these predictors only explain $\approx 1.75\%$ of the variation in the response. Thus, it follows that factors related to a student's own financial background are not necessarily very strong indicators of median earnings 10 years after graduation.

Code

```r
################################ Question 2 ###################################

## Importing Data (with changes made in Assignment 4)
diabetes <- read.csv("/Users/antonellabasso/Desktop/PHP2650/DATA/diabetes.csv")

# factorizing response variable
diabetes$early_readmit <- as.factor(diabetes$early_readmit)

# variable summary
names(diabetes)
CreateTableOne(data=diabetes)
```

```r
## Splitting the Data (75/25)
set.seed(47)

# 0.25 sample of row indices
test <- sample(1:nrow(diabetes), floor(0.25*nrow(diabetes)), replace=FALSE)

# test set
diabetes_test <- diabetes[test, ]

# train set
diabetes_train <- diabetes[-test, ]
```

```r
## Random Forest Models
set.seed(47)

# weights for imbalanced data (w_i=n_samples/n_samples_i):
w_0 <- round(nrow(diabetes_train)/sum(diabetes_train$early_readmit==0), 1) # 1.1
w_1 <- round(nrow(diabetes_train)/sum(diabetes_train$early_readmit==1), 1) # 11

# varying parameters
mtry <- seq(7, 13, 1) # all possible values of `mtry`
max_depth <- seq(4, 10, 2) # all possible values of `max.depth`

# vectors for storing values for performance eval
values <- c()
oob_error <- c()
var_imp <- c()
false_neg <- c()
true_pos <- c()
sensitivity <- c()
specificity <- c()
accuracy <- c()
auc_value <- c()

# all possible combinations of each with other specified parameters
for (i in mtry){
  for (j in max_depth){
    # values for tracking
    vals <- c(i, j)

    # model
    rf <- ranger(early_readmit ~
                   encounter_type + race_group + age_group + discharge_to_home +
                   admission_source + time_in_hospital + med_specialty +
                   num_lab_procedures + number_emergency + number_inpatient +
                   number_diagnoses + diabetesMed + diag_1_cat,
                 data=diabetes_train, importance="impurity", num.trees=100,
                 min.node.size=50, class.weights=c("0"=w_0, "1"=w_1),
                 mtry=i, max.depth=j)

    # variable importance
    rf_vi <- rf$variable.importance
    # predictions using test data
    rf_pred <- predict(rf, diabetes_test, type="response")$predictions
    # confusion matrix
    rf_cm <- confusionMatrix(rf_pred, diabetes_test$early_readmit, positive="1")

    # metrics for evaluating model performance
    oob <- rf$prediction.error # OOB error
    miv <- names(rf_vi[which(rf_vi==max(rf_vi))]) # variable with most importance
    fn <- rf_cm$table[2, 1]  # FN
    tp <- rf_cm$table[2, 2]  # TP
    sens <- rf_cm$byClass[["Sensitivity"]] # sensitivity
    spec <- rf_cm$byClass[["Specificity"]] # specificity
    acc <- rf_cm$overall[["Accuracy"]] # accuracy
    auc <- performance(prediction(as.numeric(rf_pred)-1,
                                  as.numeric(diabetes_test$early_readmit)-1),
                       measure="auc")@y.values[[1]]

    # appending values to corresponding vectors
    values <- c(values, vals)
    oob_error <- c(oob_error, oob)
    var_imp <- c(var_imp, miv)
    false_neg <- c(false_neg, fn)
    true_pos <- c(true_pos, tp)
    sensitivity <- c(sensitivity, sens)
    specificity <- c(specificity, spec)
    accuracy <- c(accuracy, acc)
    auc_value <- c(auc_value, auc)
  }
}
```

```r
## Comparing Performance of RF Models (Tables)

rfs <- as.data.frame(matrix(values, nrow=28, ncol=2, byrow=TRUE)) %>%
  mutate(oob_error, var_imp, false_neg, true_pos, sensitivity, specificity, accuracy, auc_value) %>%
  rename(mtry=V1, max.depth=V2, OOB=oob_error, MIV=var_imp, FN=false_neg, TP=true_pos,
         Sensitivity=sensitivity, Specificity=specificity, Accuracy=accuracy, AUC=auc_value)

metric <- c("OOB Error", "Sensitivity", "Specificity", "Accuracy", "AUC")

# best combination according to each (relevant) metric
best_comb <- as.data.frame(rbind(rfs[which(rfs$OOB==min(rfs$OOB)), 1:2],
                                 #rfs[which(rfs$Sensitivity==max(rfs$Sensitivity)), 1:2]
                                 c("8-11, 13", 4),
                                 rfs[which(rfs$Specificity==max(rfs$Specificity)), 1:2],
                                 rfs[which(rfs$Accuracy==max(rfs$Accuracy)), 1:2],
                                 rfs[which(rfs$AUC==max(rfs$AUC)), 1:2])) %>%
  mutate(Metric=metric)
rownames(best_comb) <- 1:5
```

```
## Plotting Change in Model Performance Over Values of `mtry` for Each `max.depth`

# OOB error
plot1 <- ggplot(data=rfs,
       aes(x=mtry, y=OOB, color=as.factor(max.depth))) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks=7:13) +
  labs(title="OOB Error Over Values of `mtry` for Each `max.depth`",
       color="max.depth")

# accuracy
plot2 <- ggplot(data=rfs,
       aes(x=mtry, y=Accuracy, color=as.factor(max.depth))) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks=7:13) +
  labs(title="Accuracy Over Values of `mtry` for Each `max.depth`",
       color="max.depth")

# AUC
plot3 <- ggplot(data=rfs,
       aes(x=mtry, y=AUC, color=as.factor(max.depth))) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks=7:13) +
  labs(title="AUC Score Over Values of `mtry` for Each `max.depth`",
       color="max.depth")
```

```
## Checking for Overfitting

# function to plot auc for train and test data over values of max_depth
auc_compare_fun <- function(mtry){
  # list for storing data frames
  df_list <- vector(mode="list", length=length(mtry))

  for (i in 1:length(mtry)){
    # auc vectors
    auc_rf_tests <- c()
    auc_rf_trains <- c()

    for (j in max_depth){
      # model
      rf <- ranger(early_readmit ~
                     encounter_type + race_group + age_group + discharge_to_home +
                     admission_source + time_in_hospital + med_specialty +
                     num_lab_procedures + number_emergency + number_inpatient +
                     number_diagnoses + diabetesMed + diag_1_cat,
```

```r
                   data=diabetes_train, importance="impurity", num.trees=100,
                   min.node.size=50, class.weights=c("0"=w_0, "1"=w_1),
                   mtry=mtry[i], max.depth=j)

      # predictions for test and train data
      rf_test <- predict(rf, diabetes_test, type="response")$predictions
      rf_train <- predict(rf, diabetes_train, type="response")$predictions

      # corresponding auc values
      auc_test <- performance(prediction(as.numeric(rf_test)-1,
                                         as.numeric(diabetes_test$early_readmit)-1),
                              measure="auc")@y.values[[1]]
      auc_train <- performance(prediction(as.numeric(rf_train)-1,
                                          as.numeric(diabetes_train$early_readmit)-1),
                               measure="auc")@y.values[[1]]

      auc_rf_tests <- c(auc_rf_tests, auc_test)
      auc_rf_trains <- c(auc_rf_trains, auc_train)
    }

    # data frame for plotting
    df <- as.data.frame(cbind(mt=rep(mtry[i], length(max_depth)),
                              max_depth=max_depth,
                              auc_test=auc_rf_tests,
                              auc_train=auc_rf_trains))

    # list of data frames
    df_list[[i]] <- df
  }

  # list for storing plots
  plots <- vector(mode="list", length=length(df_list))

  for (i in 1:length(df_list)){
    df <- df_list[[i]]

    # plotting auc values for test and train sets
    plot <- ggplot(df) +
      geom_line(aes(x=max_depth, y=auc_test), color="blue") +
      geom_line(aes(x=max_depth, y=auc_train), color="red") +
      labs(subtitle=paste0("mtry = ", df$mt[1]), x="max.depth", y="AUC") +
      annotate("text", x=10, y=max(df$auc_test), label="Test", color="blue") +
      annotate("text", x=10, y=max(df$auc_train), label="Train", color="red")

    # list of plots
    plots[[i]] <- plot
  }

  return(plots)
}
```

```r
## Plotting AUC Scores for Train and Test Data Over Values of `max.depth` and `mtry`
auc_compare_fun(mtry=mtry) # model overfits for large values of `max.depth` (>6)
```

```r
## Random Forest Model Performance for Each Value of `mtry` With  `max.depth`=8
rfs_md8 <- rfs[which(rfs$max.depth==8), -c(2, 4, 5, 6)]
rownames(rfs_md8) <- 1:7
```

```r
## Balanced Train Data
set.seed(47)

# sampling train data
class_0 <-  sample(as.numeric(rownames(diabetes_train[which(diabetes_train$early_readmit==0), ])),
                  1000, replace=FALSE)
class_1 <- sample(as.numeric(rownames(diabetes_train[which(diabetes_train$early_readmit==1), ])),
                  1000, replace=FALSE)
balanced_rows <- c(class_0, class_1)

# balanced train set
diabetes_train_b <- diabetes_train[which(rownames(diabetes_train) %in% balanced_rows), ]
```

```r
## Plotting OOB Error for Different Values of `num.trees` - ranger
set.seed(47)

oobs <- c()
for (i in 1:200){
  rf <- ranger(early_readmit ~
                 encounter_type + race_group + age_group + discharge_to_home +
                 admission_source + time_in_hospital + med_specialty +
                 num_lab_procedures + number_emergency + number_inpatient +
                 number_diagnoses + diabetesMed + diag_1_cat,
               data=diabetes_train, importance="impurity", num.trees=i,
               min.node.size=50, class.weights=c("0"=w_0, "1"=w_1),
               mtry=8, max.depth=8, write.forest=FALSE)
  oob <- rf$prediction.error
  oobs <- c(oobs, oob)
}
```

```r
# plot
oob_plot <- as.data.frame(cbind(num.trees=1:200, OOB=oobs))
ggplot(oob_plot, aes(x=num.trees, y=OOB)) +
  geom_line()
```

```r
## Plotting OOB Error for Different Values of `num.trees` - randomForest
set.seed(47)

rF <- randomForest(as.factor(early_readmit) ~ encounter_type + race_group + age_group +
                     discharge_to_home + admission_source + time_in_hospital + med_specialty +
                     num_lab_procedures + number_emergency + number_inpatient +
                     number_diagnoses + diabetesMed + diag_1_cat,
                   diabetes_train_b, mtry=8, importance=TRUE,
                   proximity=FALSE, ntree=200)

# plots
plot(rF,
     main=" ")
legend("topright",
       legend=c("Positive Pred. = 1", "Negative Pred. = 0", "Total OOB"),
       col=c("green", "red", "black"),
       lty=c(2, 2, 1), cex=0.8)

plot(1:200, rF$err.rate[, 1],
     type="line",
     main=" ",
     xlab="Number of Trees (ntree)",
     ylab="OOB Error (%)")
#abline(v=c(100, 280), col=c("red2", "blue"), lty=c(2, 2), lwd=c(2, 2))
#legend("topright",
       #legend=c("ntree = 100", "ntree = 280"),
       #col=c("red", "blue"), lty=2:2, cex=0.8)


# no max depth param
```

```r
## Final Random Forest Models - (mtry=8, max.depth=8, num.trees=)

# Approach 1: Weights
d_rf <- ranger(early_readmit ~
                 encounter_type + race_group + age_group + discharge_to_home +
                 admission_source + time_in_hospital + med_specialty +
                 num_lab_procedures + number_emergency + number_inpatient +
                 number_diagnoses + diabetesMed + diag_1_cat,
               data=diabetes_train, importance="impurity", num.trees=100,
               min.node.size=50, class.weights=c("0"=w_0, "1"=w_1),
               mtry=8, max.depth=8)

# Approach 2: Balanced Subset of Train Data
d_rf_b <- ranger(early_readmit ~
                   encounter_type + race_group + age_group + discharge_to_home +
                   admission_source + time_in_hospital + med_specialty +
                   num_lab_procedures + number_emergency + number_inpatient +
                   number_diagnoses + diabetesMed + diag_1_cat,
                 data=diabetes_train_b, importance="impurity", num.trees=100,
                 min.node.size=50, mtry=8, max.depth=8)
```

```r
## Evaluating and Comparing Model Performance

#variable importance
d_rf_vi <- d_rf$variable.importance
d_rf_b_vi <- d_rf_b$variable.importance
# predictions using test data
d_rf_pred <- predict(d_rf, diabetes_test, type="response")$predictions
d_rf_b_pred <- predict(d_rf_b, diabetes_test, type="response")$predictions
# confusion matrix
d_rf_cm <- confusionMatrix(d_rf_pred, diabetes_test$early_readmit, positive="1")
d_rf_b_cm <- confusionMatrix(d_rf_b_pred, diabetes_test$early_readmit, positive="1")

# metrics for rf with imbalanced train data
d_rf_oob <- d_rf$prediction.error # OOB error
d_rf_miv <- names(d_rf_vi[which(d_rf_vi==max(d_rf_vi))]) # variable with most importance
d_rf_fn <- d_rf_cm$table[2, 1] # FN
d_rf_tp <- d_rf_cm$table[2, 2] # TP
d_rf_sens <- d_rf_cm$byClass[["Sensitivity"]] # sensitivity
```

```
d_rf_spec <- d_rf_cm$byClass[["Specificity"]] # specificity
d_rf_acc <- d_rf_cm$overall[["Accuracy"]] # accuracy
d_rf_auc <- performance(prediction(as.numeric(d_rf_pred)-1,
                                   as.numeric(diabetes_test$early_readmit)-1),
                        measure="auc")@y.values[[1]] # AUC

# metrics for rf with balanced train data
d_rf_b_oob <- d_rf_b$prediction.error # OOB error
d_rf_b_miv <- names(d_rf_b_vi[which(d_rf_b_vi==max(d_rf_b_vi))]) # variable with most importance
d_rf_b_fn <- d_rf_b_cm$table[2, 1] # FN
d_rf_b_tp <- d_rf_b_cm$table[2, 2] # TP
d_rf_b_sens <- d_rf_b_cm$byClass[["Sensitivity"]] # sensitivity
d_rf_b_spec <- d_rf_b_cm$byClass[["Specificity"]] # specificity
d_rf_b_acc <- d_rf_b_cm$overall[["Accuracy"]] # accuracy
d_rf_b_auc <- performance(prediction(as.numeric(d_rf_b_pred)-1,
                                   as.numeric(diabetes_test$early_readmit)-1),
                        measure="auc")@y.values[[1]] # AUC

# data frame
rf_comp_df <- as.data.frame(cbind(Approach=c("Class Weights", "Balanced Train Set"),
                                  OOB=c(d_rf_oob, d_rf_b_oob),
                                  MIV=c(d_rf_miv, d_rf_b_miv),
                                  FN=c(d_rf_fn, d_rf_b_fn),
                                  TP=c(d_rf_tp, d_rf_b_tp),
                                  Sensitivity=c(d_rf_sens, d_rf_b_sens),
                                  Specificity=c(d_rf_spec, d_rf_b_spec),
                                  Accuracy=c(d_rf_acc, d_rf_b_acc),
                                  AUC=c(d_rf_auc, d_rf_b_auc)))
for (i in c(2, 4:9)){
  rf_comp_df[, i] <- as.numeric(rf_comp_df[, i])
}
```

```
################################ Question 3 ###################################

## Importing Data
college_scorecard <- read.csv("/Users/antonellabasso/Desktop/PHP2650/DATA/college_scorecard.csv")

## EDA
# general overview
names(college_scorecard) # 66 variables
dim(college_scorecard) # size = 3136 x 66
str(college_scorecard) # structure
head(college_scorecard)

# variable summary
CreateTableOne(data=college_scorecard[, -c(1)])

# length of unique values by variable
len_unique_vals <- as.data.frame(apply(college_scorecard, 2, function(x) length(unique(x)))) %>%
  rename("unique"=1)
len_unique_vals <- len_unique_vals %>%
  mutate("var"=rownames(len_unique_vals))
rownames(len_unique_vals) <- 1:nrow(len_unique_vals)
```

```r
# binary variables
bin_vars <- len_unique_vals[which(len_unique_vals$unique <= 2), ]
# positive (1) values for each binary variable
apply(college_scorecard[, which(colnames(college_scorecard) %in% bin_vars$var)], 2, function(x) sum(x))
```

```r
## Response Variable Density

# response
hist(college_scorecard$MD_EARN_WNE_P10/1000,
     probability=TRUE, col="lightblue", breaks=30,
     main="Response Variable Density",
     xlab="Median Salary 10 Years Post-Graduation (in thousands)",
     sub="In Thousands")
lines(density(college_scorecard$MD_EARN_WNE_P10/1000), col="red")

# scaled/centered response
scaled_response <- scale(college_scorecard$MD_EARN_WNE_P10,
                         center=mean(college_scorecard$MD_EARN_WNE_P10),
                         scale=sd(college_scorecard$MD_EARN_WNE_P10))
hist(scaled_response,
     probability=TRUE, col="lavender", breaks=30,
     main="Centered Response Variable Density",
     sub="Centered at Mean, Scaled by SD",
     xlab="Median Salary 10 Years Post-Graduation")
lines(density(scaled_response), col="red")
```

```r
# log-transformed response
hist(log(college_scorecard$MD_EARN_WNE_P10),
     probability=TRUE, col="thistle2", breaks=30,
     main="Log-Transformed Response Variable Density",
     xlab="log ( Median Salary 10 Years Post-Graduation )")
lines(density(log(college_scorecard$MD_EARN_WNE_P10)), col="red")
```

```r
## PART 1: RESPONSE VARIABLE


## Transforming Data Into Model Matrix

# model matrix not including institution name column (1)
  # excludes target variable and includes an intercept
cs_mat <- model.matrix(MD_EARN_WNE_P10~., college_scorecard[, -1])

# removing consequent intercept column (1)
cs_mat <- cs_mat[, -1]
```

```r
## Splitting the Data (75/25)
set.seed(47)

# 0.75 sample of row indices
train <- sample.int(nrow(college_scorecard), 0.75*nrow(college_scorecard), replace=FALSE)

# test sets
cs_test_x <- cs_mat[-train, ]
cs_test_y <- college_scorecard$MD_EARN_WNE_P10[-train]
```

```
# train sets
cs_train_x <- cs_mat[train, ]
cs_train_y <- college_scorecard$MD_EARN_WNE_P10[train]
```

```
## Standardizing Data (center=mean, scale=sd)

# getting means and sds for scaling
train_means <- apply(cs_train_x, 2, mean) # can also use colMeans(train_x)
train_sds <- apply(cs_train_x, 2, sd)

# scaling
cs_train_x <- sweep(sweep(cs_train_x, 2L, train_means), 2, train_sds, "/")
cs_test_x <- sweep(sweep(cs_test_x, 2L, train_means), 2, train_sds, "/")
```

```
### Feed-forward Neural Network 1: Response
# (done in Google Colab)
set.seed(47)

## Model Architecture
  # including layer dropout to minimize overfitting (due to noise)
    # randomly drops 20% of nodes in each layer
  # including batch normalization to minimize overfitting
cs_dnn <- keras_model_sequential() %>%
  layer_dense(units=ncol(cs_train_x)/2, # first hidden layer (size=ncol/2)
              activation="relu", # first activation function
              input_shape=ncol(cs_train_x)) %>% # input layer (size=ncol)
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=16, # second hidden layer (size=ncol/4)
              activation="relu") %>%  # second activation function
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=8, # third hidden layer (size=ncol/8)
              activation="relu") %>% # third activation function
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=1) # implied output function/activation="identity"
summary(cs_dnn) # summary

## Back-propagation
cs_dnn %>%
  compile(optimizer="rmsprop", # RMSProp
          loss="mse", # mean squared error
          metrics=c("mae")) # mean absolute error

## Training Model ("Learning")
  # stops if there is no improvement after 5 epochs (early stopping)
cs_learn <- cs_dnn %>%
  fit(x=cs_train_x, # train features
      y=cs_train_y, # train response
      epochs=100, # number of times train data is seen by the algorithm
      batch_size=64, # sample/batch size for mini-batch SGD
      validation_split=0.2, # CV hold out set
      verbose=FALSE, # excludes live updates of loss function
      callbacks=callback_early_stopping(patience=4)) # early stopping

## Plotting Validation Metric
plot(cs_learn)

## Final NN Model, After Tuning Parameters
cs_dnn <- keras_model_sequential() %>%
  # architecture
  layer_dense(units=ncol(cs_train_x), # 64
              activation="relu",
              input_shape=ncol(cs_train_x)) %>% # 64
  layer_dense(units=ncol(cs_train_x), # 64
              activation="relu") %>%
  layer_dense(units=ncol(cs_train_x)/2, # 32
              activation="relu") %>%
  layer_dense(units=ncol(cs_train_x)/2, # 32
              activation="relu") %>%
  layer_dense(units=1) %>% # 1
  # backpropagation
  compile(optimizer="rmsprop",
          loss="mse",
          metrics=c("mae"))  %>%
  # training
  fit(x=cs_train_x, # cs_learn
      y=cs_train_y,
      epochs=40,
      batch_size=100,
      validation_split=0.2,
      callbacks=callback_early_stopping(patience=4))

## Plotting Validation Metric
plot(cs_learn)

## Model Performance & Predictions
set.seed(47)
cs_dnn %>% evaluate(cs_test_x, cs_test_y, verbose=FALSE)
cs_pred_probs <- predict(cs_dnn, cs_test_x)
#cbind(cs_pred_probs, cs_test_y)
```

```
## Performance & Prediction Tables

# before and after tuning parameters
ba <- c("Before", "After")
loss <- c(1426709504, 41050780)
mae <- c(35837.3125, 4370.4873)
pt <- as.data.frame(cbind(loss, mae)) %>%
  rename("Loss"=loss, "MAE"=mae)
rownames(pt) <- ba

# first 10 observed vs. predicted responses
cs_pred_10 <- c(34234.37, 31934.74, 26417.44, 26235.96, 39618.32,
                25096.07, 29677.95, 40391.52, 30069.89, 26403.80)
pred_10 <- as.data.frame(cbind(cs_test_y[1:10], cs_pred_10)) %>%
  rename("Observed"=1, "Predicted"=2) %>%
  mutate("% Error"=((Predicted-Observed)/Observed)*100)
```

```
## PART 2: LOG-RESPONSE VARIABLE

set.seed(47) #  using the same train and test sets
# cs_test2_y[1:10]=log(cs_test_y[1:10])
# exp(cs_test2_y[1:10])=cs_test_y[1:10]

## Transforming Data Into Model Matrix

# log-transformed response
college_scorecard2 <- college_scorecard
college_scorecard2$MD_EARN_WNE_P10 <- log(college_scorecard$MD_EARN_WNE_P10)

# model matrix not including institution name column (1)
  # excludes target variable and includes an intercept
cs_mat2 <- model.matrix(MD_EARN_WNE_P10~., college_scorecard2[, -1])
#could we cs_mat2 <- model.matrix(log(MD_EARN_WNE_P10)~., college_scorecard[, -1]) ?

# removing consequent intercept column (1)
cs_mat2 <- cs_mat2[, -1]


## Splitting the Data (75/25)

# 0.75 sample of row indices
train2 <- sample.int(nrow(college_scorecard2), 0.75*nrow(college_scorecard2), replace=FALSE)

# test sets
cs_test2_x <- cs_mat2[-train2, ]
cs_test2_y <- college_scorecard2$MD_EARN_WNE_P10[-train2]

# train sets
cs_train2_x <- cs_mat2[train2, ]
cs_train2_y <- college_scorecard2$MD_EARN_WNE_P10[train2]


## Standardizing Data (center=mean, scale=sd)

# getting means and sds for scaling
train2_means <- apply(cs_train2_x, 2, mean)
train2_sds <- apply(cs_train2_x, 2, sd)

# scaling
cs_train2_x <- sweep(sweep(cs_train2_x, 2L, train2_means), 2, train2_sds, "/")
cs_test2_x <- sweep(sweep(cs_test2_x, 2L, train2_means), 2, train2_sds, "/")
```

```
### Feed-forward Neural Network 2: Log-Transformed Response
# (done in Google Colab)
set.seed(47)

## Model Architecture
  # including layer dropout to minimize overfitting (due to noise)
    # randomly drops 20% of nodes in each layer
```

```r
  # including batch normalization to minimize overfitting
cs_dnn2 <- keras_model_sequential() %>%
  layer_dense(units=ncol(cs_train2_x)/2, # first hidden layer (size=ncol/2)
              activation="relu", # first activation function
              input_shape=ncol(cs_train2_x)) %>% # input layer (size=ncol)
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=16, # second hidden layer (size=ncol/4)
              activation="relu") %>%  # second activation function
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=8, # third hidden layer (size=ncol/8)
              activation="relu") %>% # third activation function
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=1) # implied output function/activation="identity"
summary(cs_dnn2) # summary

## Back-propagation
cs_dnn2 %>%
  compile(optimizer="rmsprop", # RMSProp
          loss="mse", # mean squared error
          metrics=c("mae")) # mean absolute error

## Training Model ("Learning")
  # stops if there is no improvement after 5 epochs (early stopping)
cs_learn2 <- cs_dnn2 %>%
  fit(x=cs_train2_x, # train features
      y=cs_train2_y, # train response
      epochs=100, # number of times train data is seen by the algorithm
      batch_size=64, # sample/batch size for mini-batch SGD
      validation_split=0.2, # CV hold out set
      verbose=FALSE, # excludes live updates of loss function
      callbacks=callback_early_stopping(patience=4)) # early stopping

## Plotting Validation Metric
plot(cs_learn2)

## Final NN Model, After Tuning Parameters
cs_dnn2 <- keras_model_sequential() %>%
  # architecture
  layer_dense(units=ncol(cs_train2_x), # 64
              activation="relu",
              input_shape=ncol(cs_train2_x)) %>% # 64
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=16,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=8,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=1) %>%
  # backpropagation
  compile(optimizer="rmsprop",
          loss="mse",
          metrics=c("mae"))  %>%
  # training
  fit(x=cs_train2_x, # cs_learn2
      y=cs_train2_y,
      epochs=50,
      batch_size=50,
      validation_split=0.2,
      callbacks=callback_early_stopping(patience=4))

## Plotting Validation Metric
plot(cs_learn2)

## Model Performance & Predictions
set.seed(47)
cs_dnn2 %>% evaluate(cs_test2_x, cs_test2_y, verbose=FALSE)
cs_pred_probs2 <- predict(cs_dnn2, cs_test2_x)
#cbind(cs_pred_probs2, cs_test2_y)
```

```
## Performance & Prediction Tables

# before and after tuning parameters
loss2 <- c(0.0355498, 0.0197238)
mae2 <- c(0.1462143, 0.1007187)
pt2 <- as.data.frame(cbind(loss2, mae2)) %>%
  rename("Loss"=loss2, "MAE"=mae2)
rownames(pt2) <- ba

# first 10 observed vs. predicted responses
cs_pred2_10 <- c(10.495963, 10.287291, 10.238688, 10.188322, 10.528452,
                 10.282755, 10.245202, 10.750225, 10.459530, 10.180101)
pred2_10 <- as.data.frame(cbind(cs_test2_y[1:10], cs_pred2_10)) %>%
  rename("Observed"=1, "Predicted"=2) %>%
  mutate("% Error"=((Predicted-Observed)/Observed)*100)
```

```
### Comparing Feed-forward Neural Networks

## Model 1:

# MAE
 mean(abs(cs_pred_probs-cs_test_y)) # 4370.4873
# MA%E <- mean absolute percentage error for relative comparison
mean(abs((cs_pred_probs-cs_test_y)/cs_test_y)*100) # 12.7544
# LM Approximation
lm1 <- lm(cs_pred_probs~cs_test_y)
summary(lm1)
# Regression Plot
plot(lm1, main="NN Model 1: Response",
     xlab="Observed", ylab="Predicted")
abline(lm(cs_pred_probs~cs_test_y),
       col="red", lwd=2.5, lty=2)

## Model 2:

# MAE
mean(abs(cs_pred_probs2-cs_test2_y)) # 0.1007
# MA%E
mean(abs((cs_pred_probs2-cs_test2_y)/cs_test2_y)*100) # 0.9607
# LM Approximation
lm2<- lm(cs_pred_probs2~cs_test2_y)
summary(lm2)
# Regression Plot
plot(lm2, main="NN Model 2: Log Response",
     xlab="Observed", ylab="Predicted")
abline(lm(cs_pred_probs2~cs_test2_y),
       col="red", lwd=2.5, lty=2)
```

```
## Model Comparinson Tables

models <- c("Model 1: Response", "Model 2: Log Response")
MAE <- c(4370.4873, 0.1007)
MAPE <- c(12.7544, 0.9607)
RSE <- c(5943, 0.1218)
RSQ <- c(0.7154, 0.8086)

mc <- as.data.frame(cbind(MAE, MAPE, RSE, RSQ)) %>%
  rename("MA%E"=MAPE, "R2"=RSQ)
rownames(mc) <- models
```

```
## Residuals vs. Fitted
plot(lm(MD_EARN_WNE_P10~., college_scorecard[, -1]), which=1) # response
plot(lm(log(MD_EARN_WNE_P10)~., college_scorecard[, -1]), which=1) # log-response

## Q-Q
plot(lm(MD_EARN_WNE_P10~., college_scorecard[, -1]), which=2) # response
plot(lm(log(MD_EARN_WNE_P10)~., college_scorecard[, -1]), which=2) # log-response
```

```
### Feed-forward Neural Network 2b: Log-Transformed Response
###(Removing Individual Financial Background)

## Data Excluding Relevant Predictors
sfb <- c("PELL_EVER", "MEDIAN_HH_INC", "POVERTY_RATE", "UNEMP_RATE")
cs_train3_x <- cs_train2_x[, !(colnames(cs_train2_x) %in% sfb)]
cs_test3_x <- cs_test2_x[, !(colnames(cs_test2_x) %in% sfb)]
cs_train3_y <- cs_train2_y
cs_test3_y <- cs_test2_y

## NN Model
cs_dnn3 <- keras_model_sequential() %>%
  # architecture
  layer_dense(units=ncol(cs_train3_x), # 60
              activation="relu",
              input_shape=ncol(cs_train3_x)) %>% # 60
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=16,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=8,
              activation="relu") %>%
  layer_dropout(rate=0.2) %>% # layer dropout
  layer_batch_normalization() %>% # batch normalization
  layer_dense(units=1) %>%
  # backpropagation
  compile(optimizer="rmsprop",
          loss="mse",
          metrics=c("mae"))  %>%
  # training
  fit(x=cs_train3_x, # cs_learn3
      y=cs_train3_y,
      epochs=50,
      batch_size=50,
      validation_split=0.2,
      callbacks=callback_early_stopping(patience=4))

## Plotting Validation Metric
plot(cs_learn3)

## Model Performance & Predictions
set.seed(47)
cs_dnn3 %>% evaluate(cs_test3_x, cs_test3_y, verbose=FALSE)
cs_pred_probs3 <- predict(cs_dnn3, cs_test3_x)
#cbind(cs_pred_probs3, cs_test3_y)
mean(abs(cs_pred_probs3-cs_test3_y)) # MAE
mean(abs((cs_pred_probs3-cs_test3_y)/cs_test3_y)*100) # MA%E
```

```
## Performance & Prediction Tables

# first 10 observed vs. predicted responses
cs_pred3_10 <- c(10.516086, 10.296928, 10.374944, 10.190425, 10.465935,
                 10.300498, 10.246922, 10.689170, 10.244572, 10.302035)
pred3_10 <- as.data.frame(cbind(cs_test2_y[1:10], cs_pred2_10, cs_pred3_10)) %>%
  rename("Observed"=1, "Predicted 2"=2, "Predicted 2b"=3)
pred3_10["% Error 2"] <- ((pred3_10[, 2]-pred3_10$Observed)/pred3_10$Observed)*100
pred3_10["% Error 2b"] <- ((pred3_10[, 3]-pred3_10$Observed)/pred3_10$Observed)*100

# before and after removing predictors
loss3 <- c(0.0197, 0.0230)
mae3 <- c(0.1007, 0.1103)
mape3 <- c(0.9607, 1.0568)
rse3 <- c(0.1218, 0.1137)
rsq3 <- c(0.8086, 0.7912)
pt3 <- as.data.frame(cbind(loss3, mae3, mape3, rse3, rsq3)) %>%
  rename("Loss"=loss3, "MAE"=mae3, "MA%E"=mape3,
         "RSE"=rse3,"R2"=rsq3)
rownames(pt3) <- c("Model 2", "Model 2b")
```