COMP4211 Project

Forecast of Resource Usage by Patients in Hospitals

By MENOR Edrik Jan Valdez (20696958) (ejvmenor@connect.ust.hk)

1. Introduction and Abstract

With the trend of population ageing and worsening physical and mental disorders, demand for public health services, specifically from hospitals, is swelling. Often, the existing resources in public hospitals may not meet such swelling demand. This spawns the need for developing methods to allow efficient and effective allocation of hospital resources to cater to differing needs from hospital patients. Specifically, this includes identifying patients needing greater reliance on hospital resources, such as longer stays and treatment for medical complications.

The mission of this project is to use the patient's records from the Hospital Length of Stay Dataset to build an ML model to use a patient's physiological and health profile (excluding readmission count), to predict the length of hospital stay (by regression) and readmission count (by classification).

2. Dataset
    a. Information

The dataset used is the eponymous Hospital Length of Stay Dataset available by Kaggle. The dataset contains 100000 case records containing attributes related to patients' health and information regarding hospital stay, such as facility, entry and discharge dates, readmission count and length of stay.

This raw version of dataset has the following columns:

Name	Description	Value Range and Data Type
eid	Case ID given to the patient	1 – 100000 (integer)
vdate	Date of admission to hospital of the patient	dd/mm/yy (date)
rcount	Number of readmissions to the hospital by the patient	0, 1, 2, 3, 4, 5+ (integer, category)
gender	Gender of the patient	F, M
dialysisrenalendstage	Indicator whether the patient undergoes dialysis, a treatment for failing kidneys	0, 1 (Boolean/integer)
asthma	Indicator whether the patient suffers from asthma	0, 1 (Boolean/integer)
irondef	Indicator whether the patient suffers from iron deficiency	0, 1 (Boolean/integer)
pneum	Indicator whether the patient suffers from pneumonia	0, 1 (Boolean/integer)
substancedependence	Indicator whether the patient is dependent on substances (esp. drugs)	0, 1 (Boolean/integer)
psychologicaldisordermajor	Indicator whether the patient suffers from any psychological disorder	0, 1 (Boolean/integer)
depress	Indicator whether the patient suffers from depression	0, 1 (Boolean/integer)
psychother	Whether the patient suffers from other/special psychological disorders	0, 1 (Boolean/integer)

fibrosis	Whether the patient suffers from muscular fibrosis	0, 1 (Boolean/integer)
malnutrition	Whether the patient suffers from malnutrition	0, 1 (Boolean/integer)
hemo	Whether the patient has hemorrhage	0, 1 (Boolean/integer)
hematocrit	Percentage of red blood cells in blood composition	4.4 ~ 24.1 (float)
neutrophils	Number of neutrophils, the most common type of white blood cells	0.1 ~ 246 (float)
sodium	sodium level in blood (mEq/L)	125 ~ 151 (float)
glucose	Blood glucose level (mg/dL)	-1.01 ~ 271 (float)
bloodureanitro	Level of blood urea nitrogen (mg/dL)	1 ~ 683 (float)
creatinine	Level of creatinine (mg/dL)	0.22 ~ 2.04 (float)
bmi	Body mass index	22 ~ 38.9 (float)
pulse	Heart rate (pulses/min)	21 ~ 130 (int)
respiration	No. of breaths taken per minute	0.2 ~ 10 (float)
secondarydiagnosisnonicd9	No. of coexisting conditions of patient at time of admission	0 ~ 10 (int)
discharged	Date of a patient being discharged	dd/mm/yyyy (date)
facid	Facility ID the patient was in	A, B, C, D, E (string)
lengthofstay	The duration in days a patient is hospitalized	1~17 (int)

Table 1: Columns of Hospital Length of Stay Dataset

b. Preprocessing

First, we need to perform data cleaning and preprocessing.
From the dataset info printed on the notebook, there are no missing data.
Here are the following sequential actions:

i. Remove irrelevant columns, which are "eid", "vdate", "discharged" and "facid", since they are administrative information without anything to do with patient's health.
ii. Fully numericize the dataset by numericizing the following:
    - "gender" column: F → 0, M →1
    - "rcount" column: 5+ → 5
iii. Normalize all of the input features

.

c. Dataset partitioning

As a common practice for training ML models, we usually split the dataset into training, validation, and testing partitions. Also, we want to reduce the imbalance of our training data used for our forecasting models (i.e., length of stay prediction, readmission count prediction). It is because we do not want our forecasting models to output predictions skewed towards certain categories of patients, especially in case of underfitting.

In our case, our output labels are "rcount" (readmission count) and lengthofstay" (Length of hospitalization). We want to partition the dataset (features and labels) into partitions where each partition has a similar distribution as that of the population.

Therefore we partition the dataset as follows:
(i) The train-validation-test ratio is 0:68: 0.12: 0.2
(ii) Each partition has a similar distribution as population.

To do so, we need to examine the distributions of the values of "rcount" and "lengthofstay":

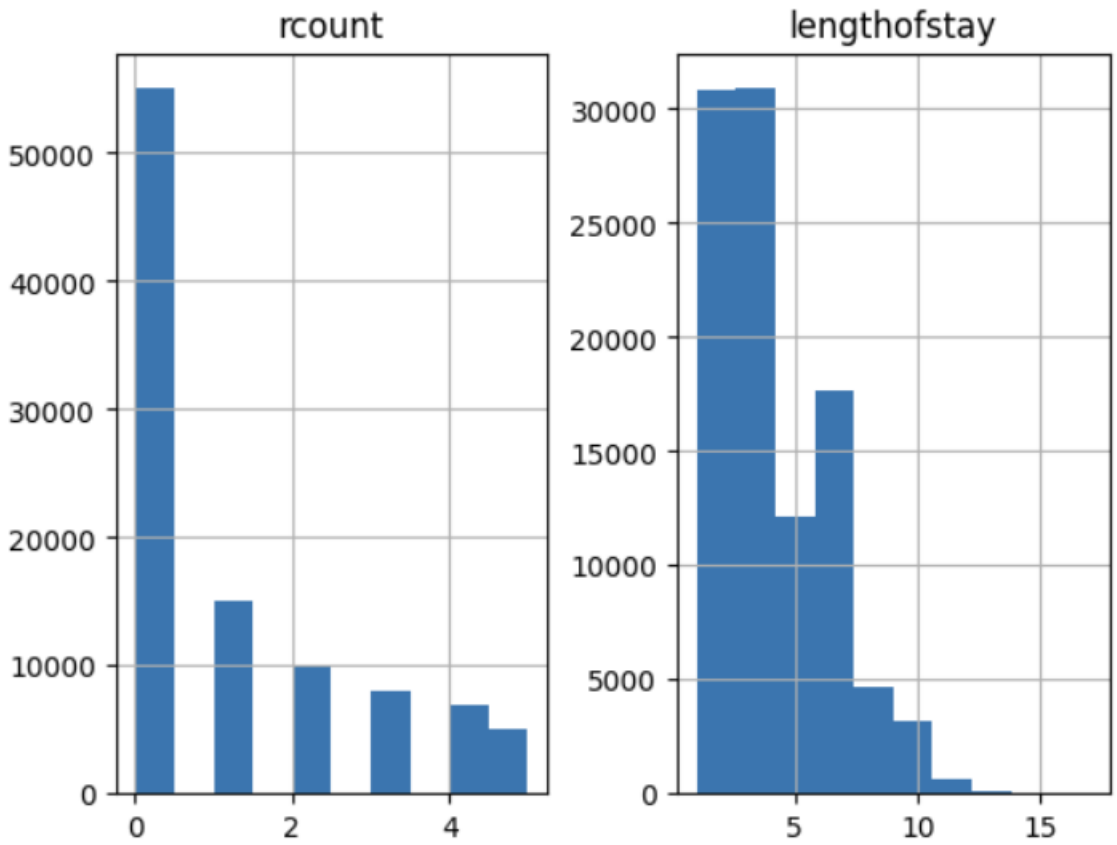


Figure 1: Histograms of values of rcount and lengthofstay

Observe that the values of the two label variables follow a similar statistical distribution. This is validated quantitatively from the correlation matrix.

	rcount	lengthofstay	secondarydiagnosisnonicd9
rcount	1.000000	0.749514	0.004233
lengthofstay	0.749514	1.000000	0.006540
secondarydiagnosisnonicd9	0.004233	0.006540	1.000000

Table 2: Correlation matrix of possible labels

In fact, the features “rcount” and “lengthofstay” have the highest absolute pairwise correlation, amongst all features, as seen from the correlation matrix for entire dataset

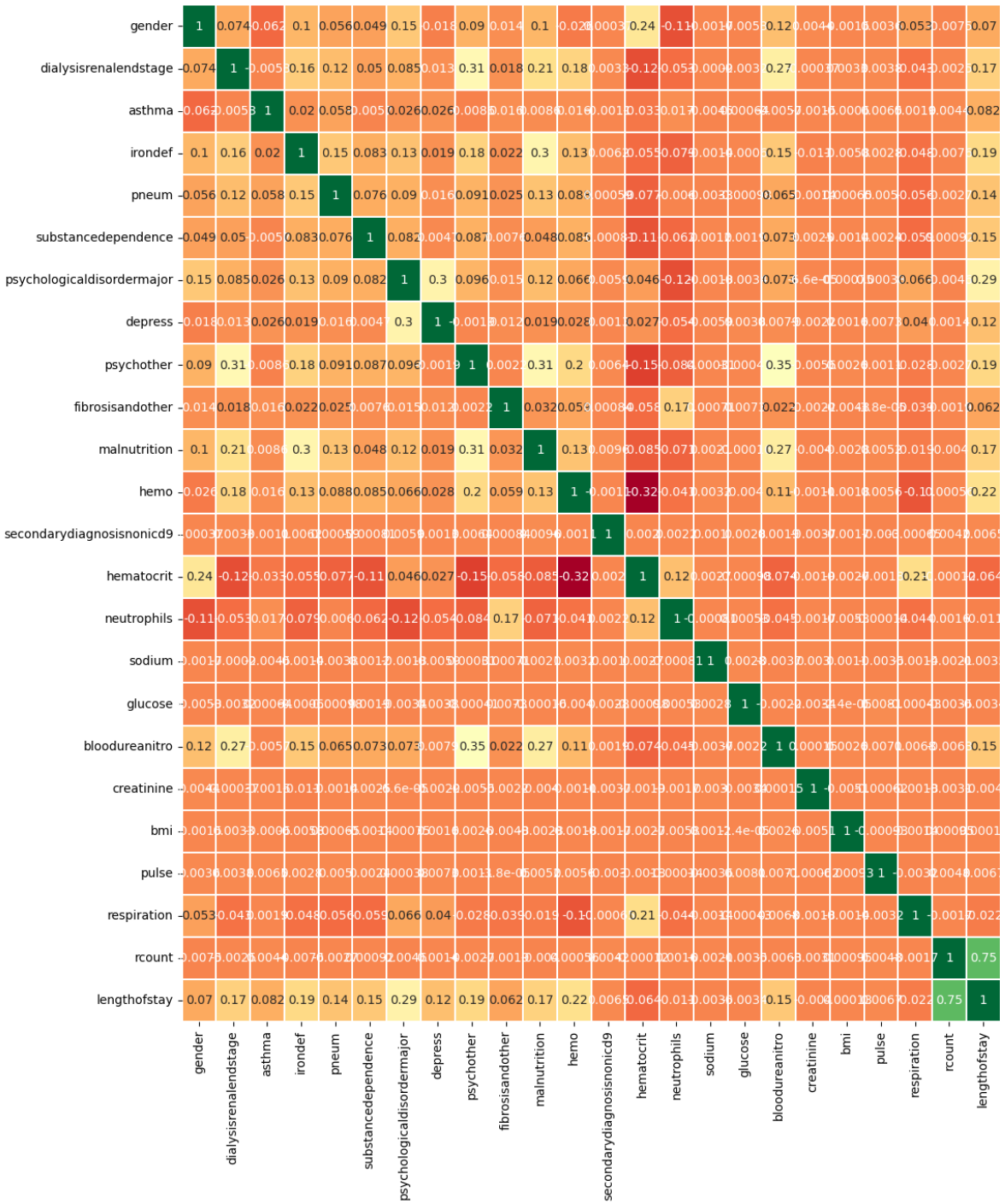


Figure 2: Correlation matrix of the entire dataset

Therefore, it suffices that we partition the model by applying stratified sampling with the column "rcount" onto the dataset with the ratio 68:12:20.

3. Methodology and implementation

This project aims to process patient's information to infer the predicted length of stay in hospitals and the readmission count. The model we design is composed of two parts: *dimensionality reduction* and *regression and classification*. Here we introduce the building blocks in their implementation methods, then we piece them back altogether.

A. Overview of components

First, define an input feature vector $X \in R^m$, containing the attributes of a patient's health.

1. Dimensionality reduction

First, we might transform vector X into $X' \in R^n$ where $m \geq n$ through a dimensionality reduction function $C: R^m \rightarrow R^n$. Sometimes $X' = X$. Here we implement C using one technique:

Autoencoder

Autoencoder is a type of feedforward neural network specifically used to reduce the dimensionality of data. It is composed of encoders, bottleneck layers and decoders.

We have a simple autoencoder in our implementation: first define an encoder $A_1: R^m \rightarrow R^h$ where $m > h$, a bottleneck layer $b \in R^h$ and a decoder $A_2: R^h \rightarrow R^n$ where $n = m$. Then $X' = A_2(A_1(X))$

In our designed model, there are two options:

1. Autoencoding
2. No dimensionality reduction.

2. Regression and classification

Now that we have transformed input vector $X' \in R^n$, we will use two methods to output the predicted length of stay and the predicted readmission count:

Deep neural network method

This method involves the construction of several feedforward neural networks, namely the common ResNet, regression layer and the classification layer.

A. Common ResNet

We will build 4 sequential small custom residual feedforward neural network blocks. There are composed of a sequence of w blocks of sequential layers $N = (L_1, L_2, \ldots, L_w, p)$, and a skip connection s.
Here, we have $L_i = (f_i, g_i, h_i)$. f_i is a fully connected layer equipped with an L2 regularizer, g_i is a batch normalization layer. h_i is a LeakyReLU activation layer with slope 0.3.
p is a fully connected layer with activation function to resize the output to the dimension of the input for skip connection. s maps $X'' \in R^n$ from the top of L_1 to the bottom of L_w. Then we have output $X'' = N(X') + X'$.

B. SLP regression and classification layers
We will build a single regression layer $RL \in R^{n/2}$ to output the predicted length of stay $y_1 = RL(X'')$, and a single classification layer $CL \in R^{n/2}$ to output the predict readmission count $y_2 = CL(X'')$. Each of these layers are equipped with an activation function, and an activity regularizer. Here we have $y_1 > 0$ and $y_2 \in \{0, 1,2,3,4,5\}$.

Decision tree method (only without dimensionality reduction)

This method involves the usage and training of decision tree models. It is used for classification and regression problems determining the decision tree internal nodes from the input features against the leaf nodes from the targets. Here we define decision trees T_1 and T_2.

T_1 is a decision tree regressor for predicting the length of stay $y_1 = T_1(X')$

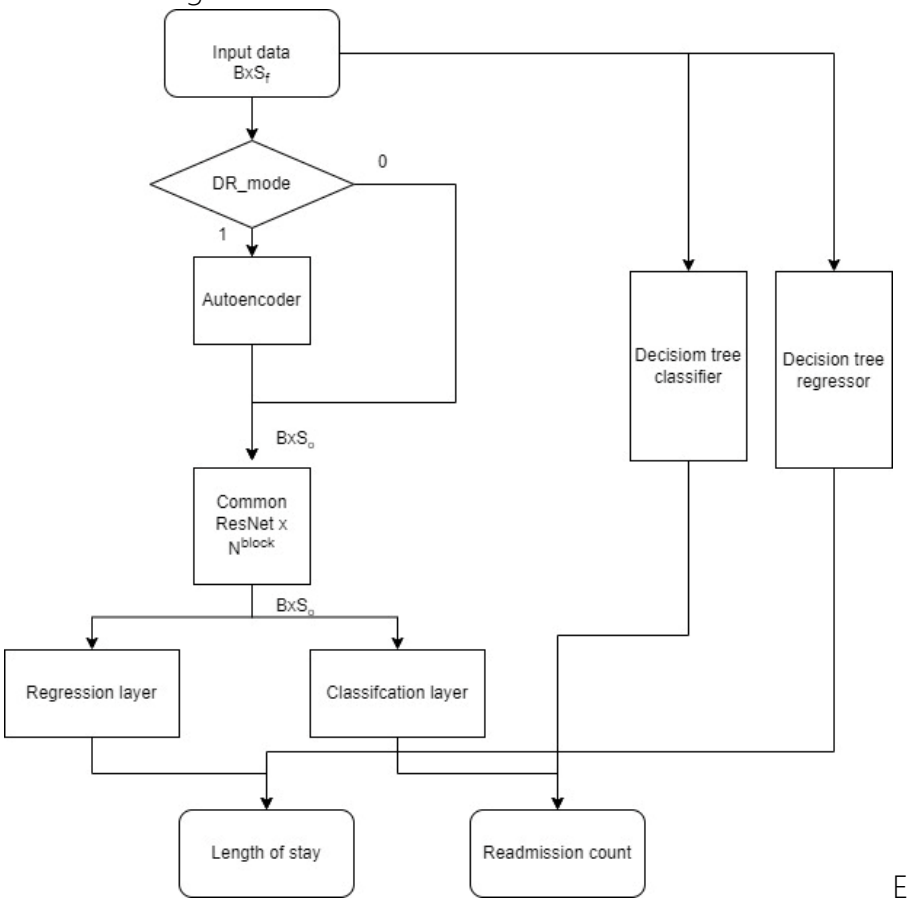
T_2 is a decision tree classifier for predicting the readmission count $y_2 = T_2(X')$

B. Overview of the model

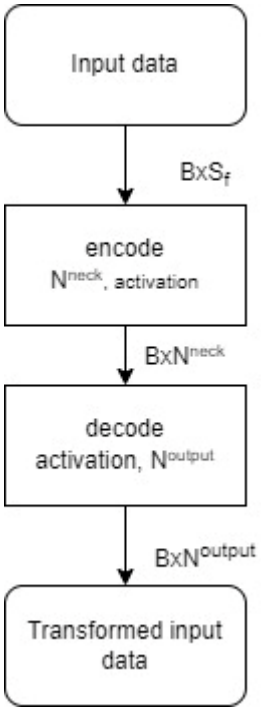
The sequential flow of the model are as follows:

1. Dimensionalty Reduction (one of a or b)
    a. Autoencoder
    b. No dimensionality reduction
2. Regression and classification section (any of a or b)
    a. Deep neural network
        i. Common ResNet
        ii. Output layers
            1. Regression layer
               Output length of stay
            2. Classification layer
               Output readmission count
    b. Decision trees (only if 1 (b) is chosen)
        i. Decision tree regressor
           Output length of stay
        ii. Decision tree classifier
           Output readmission count

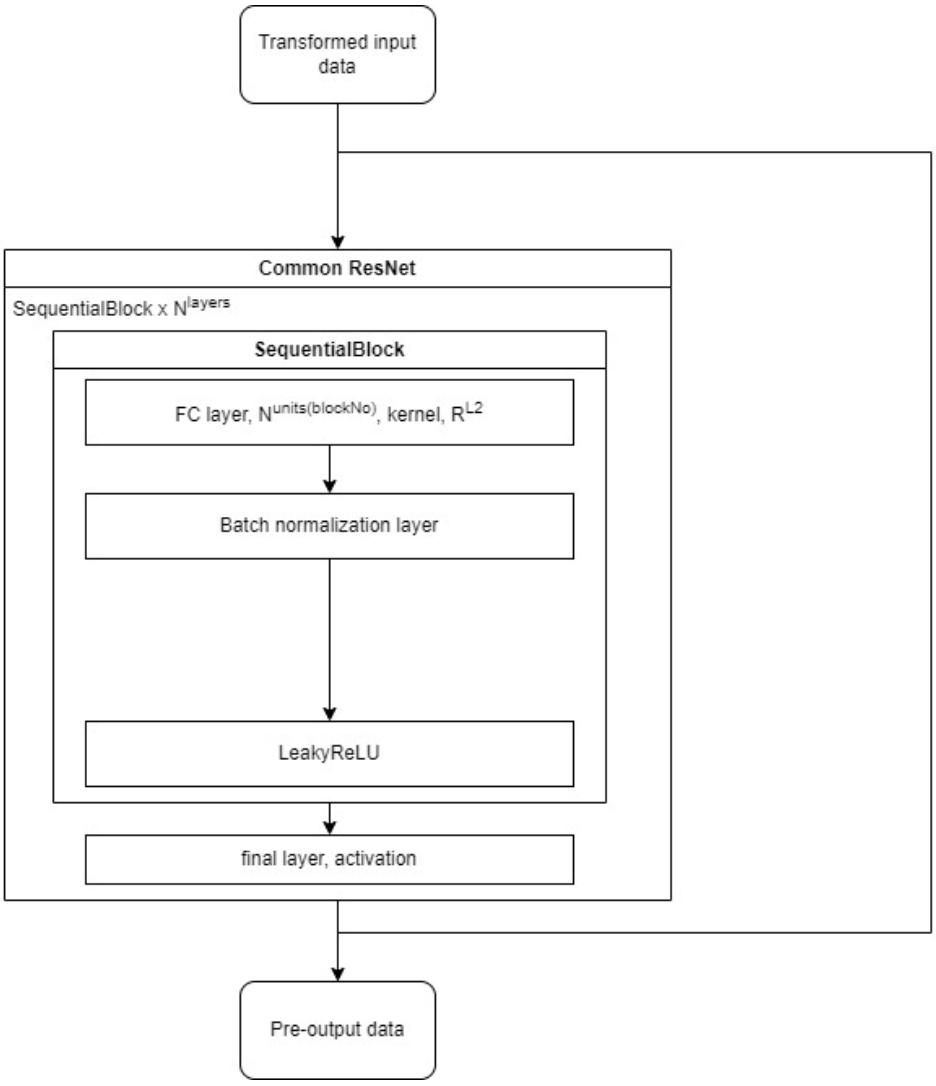
C. Model diagrams



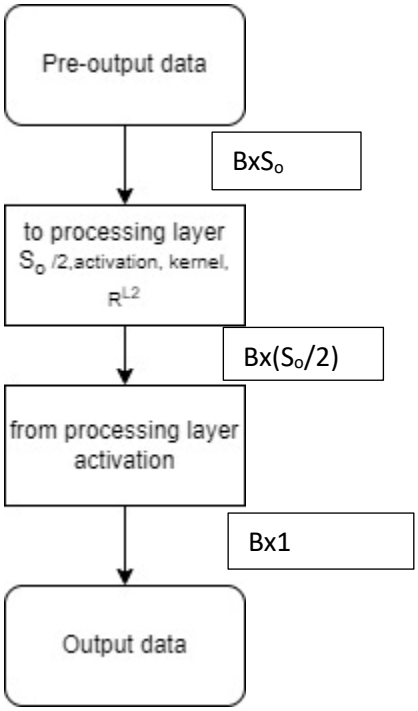
Entire model



Autoencoder



Common ResNet



Regression and classification layer

D. Notation and key hyperparameters

Symbol	Name in code	Default value	Meaning
Input and output			
B	batch_size	200	Batch size in training/testing
S_f	feature_size	22	No. of input features in original data
S_o	Input_dim	N/A	(Transformed) feature dimensions before processing output
N^{neck}	n_autoencoder_neck	10	Dimension of bottleneck layer in autoencoder
N^{output}	n_output	N/A	Dimension of output in autoencoder
Model parameters			
DR_mode	dr_toggle	0	Choice of dimensionality

			reduction method 0: No methods 1: Autoencoder
N^{layers}	no_layers	4	Number of SequentialBlock in the common ResNet
N^{block}	/	4	Number of Common ResNet blocks
N^{input}	n_input	S_o	Dimension of input and output in common ResNet
L^{layers}	n_units_layers	$[44,60,60,44]$	List of fully connected layer sizes in sequential order
$N^{units(blockNo)}$	N/A	N/A	An element in L^{layers}
activation	activation	"relu"	Common activation function in model
R^{L2}	L2_r	0.01	L2 regularization factor

Component name	Meaning
Autoencoder	A simple autoencoder
Common ResNet	A common residual FNN block
Regression layer	Layer of outputting length of stay
Classification layer	Layer outputting readmission count
SequentialBlock	Blocks of sequential layers of FC, batch normalization, dropout and activation.
Final layer	Layer in Common ResNet to resize output to input shape.
To processing layer	Function mapping input to regression/classification layer
From processing layer	Function mapping layer to final output

E. Minor functions

The following minor functions are used to finetune the intermediate outputs during neural network training:

1. Activation functions
   a. ReLU

      A non-differentiable function $f: \mathbb{R} \rightarrow [0, \infty)$ where

      $$f(x) = \begin{cases} x \; if \; x > 0 \\ 0 \; otherwise \end{cases}$$

   b. ELU

      A differentiable function $f: \mathbb{R} \rightarrow (-\alpha, \infty)$ where

      $$f(x) = \begin{cases} x \; if \; x > 0 \\ \alpha(e^x - 1) \; otherwise \end{cases}$$

      for some constant α with default value $\mathbf{1.0}$.

   c. Sigmoid

      A differentiable function $f: \mathbb{R} \rightarrow (0,1)$ where

      $$f(x) = \frac{1}{1 + e^{-x}}$$

   d. LeakyReLU

      A non-differentiable function $f: \mathbb{R} \rightarrow \mathbb{R}$ where

      $$f(x) = \begin{cases} x \; if \; x > 0 \\ \alpha x \; otherwise \end{cases}$$

      for some constant α with default value $\mathbf{0.3}$ for our implementation.

2. L2 regularizer

   A regularizer applying L2 penalty $L = Error(y, \hat{y}) + (R^{L2}\|w\|_2)$Where y is the real output, $\hat{y} = w^T x + b$ is the predicted output.

4. Setup

a. Development environment

Hardware

This project is being performed on a Google Colab machine with the following specifications:

CPU

Model name: Intel Xeon

Number of sockets:1

Number of cores per socket:1
Number of threads per core:2
Operating frequency: 2.3Ghz

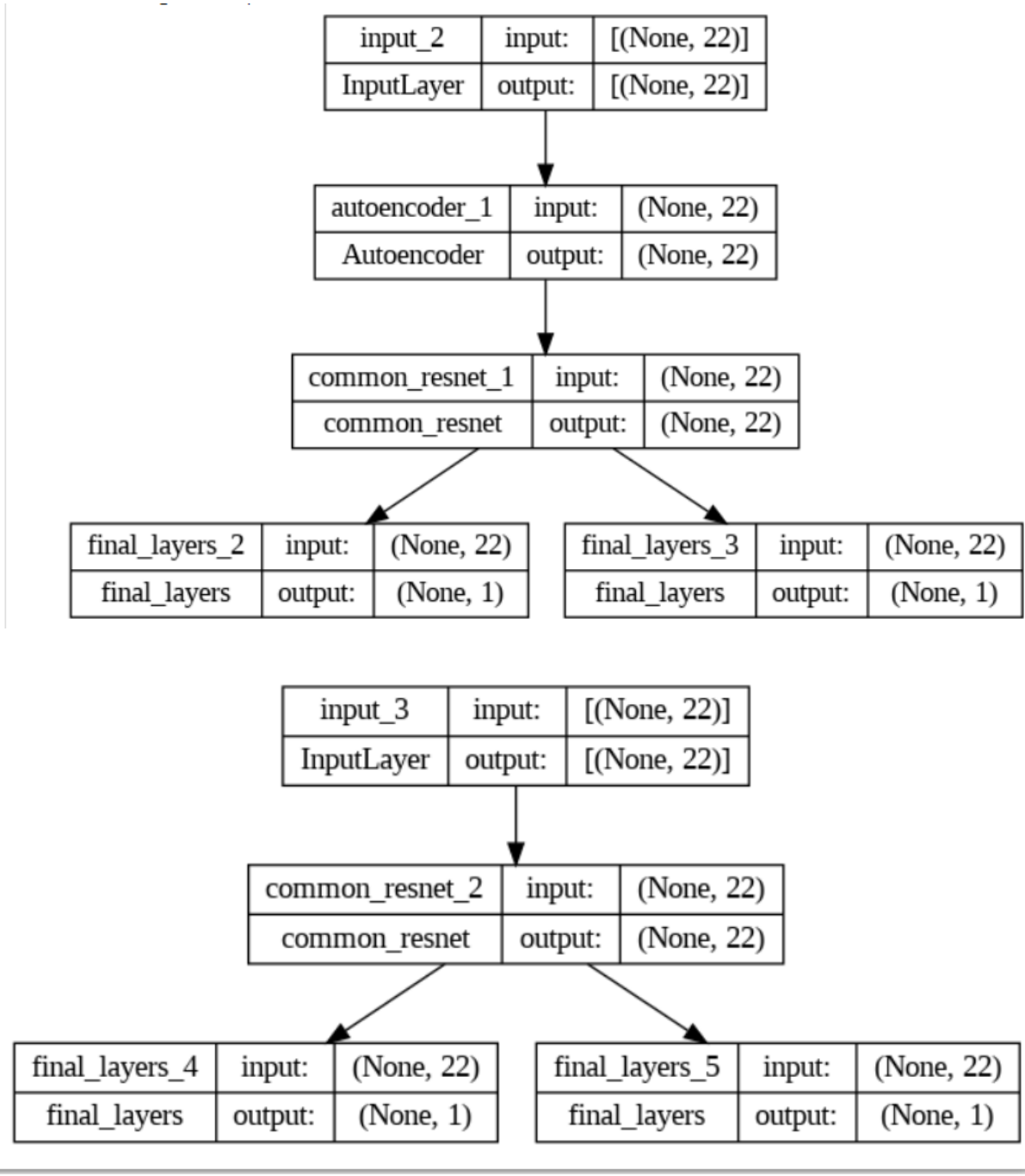
GPU
Name: Nvidia T4
VRAM size: 15GB

Software
This project is implemented in a Linux machine provided by Google Colab. These machines are preinstalled with necessary libraries:
numpy
tensorflow (comes with keras)
pandas
matplotlib
seaborn
sklearn

From these libraries, we implemented the following methods:

1. Preprocessing
   Normalization: sklearn.preprocessing.StandardScaler()
   Correlation matrix: plot of correlation matrix using seaborn
   Dataset partitioning: train_test_split
2. Neural network models
   Neural network layers: keras.layers.Layer
   Activation function: keras.activations
   L2 regularizer: keras.regularizers.L2
   Batch normalization: keras.BatchNormalization()
3. Decision Trees
   Decision tree regressor: sklearn.tree.DecisionTreeRegressor()
   Decision tree classifier: sklearn.tree.DecisionTreeClassifier()
4. Loss, metrics, and optimizer:
   Keras.optimizers, keras.losses, keras.metrics, keras.callback
   Sklearn.metrics

b. Neural network model overview with default parameters, and $N^{block} = 1$



c. Parameter and optimizer settings

Neural network training parameter settings

Here we design the following parameter sets for our training:

Parameter set 1		Parameter set 2	
Name	Value	Name	Values
DR_mode	0	DR_mode	0
N^{neck}	/	N^{neck}	/
Activation	"sigmoid"	Activation	"elu"
N^{layers}	4	N^{layers}	4
L^{layers}	[35, 35, 35, 35]	L^{layers}	[28, 28, 28, 28]
R^{L2}	0.001	R^{L2}	0.01

Parameter set 3		Parameter set 4	
Name	Value	Name	Values
DR_mode	0	DR_mode	1
N^{neck}	/	N^{neck}	10
Activation	"relu"	Activation	"sigmoid"
N^{layers}	3	N^{layers}	2
L^{layers}	[46, 46, 46]	L^{layers}	[70, 70]
R^{L2}	0.01	R^{L2}	0.001

Parameter set 5		Parameter set 6	
Name	Value	Name	Values
DR_mode	1	DR_mode	1
N^{neck}	11	N^{neck}	10
Activation	"elu"	Activation	"relu"
N^{layers}	4	N^{layers}	2
L^{layers}	[35, 35, 35, 35]	L^{layers}	[56, 56]
R^{L2}	0.01	R^{L2}	0.001

Here, we define Model $\boldsymbol{n}$ as the neural network model trained with parameter set $\boldsymbol{n}$.

Neural network training and optimizer setting

The entire neural network model will be trained in E epochs with a batch size of B. Here are their default values:

E		B	
value	15	value	200

The neural network training optimizer used will be the ADAM optimizer implemented with keras.optimizers.Adam with the following parameters:

Parameter name	value
learning_rate	5e-4
Beta_1	0.94
Beta_2	0.9994
Epsilon	5e-7

Decision tree training parameter setting

For each decision tree model implemented with sklearn.tree we have the parameters: criterion set to "squared_error" and "entropy" and split set to "best"

d. Performance evaluation

Neural network method metrics

Here the following metrics and loss functions used to train, validate, and evaluate our neural network model, with each one taking default parameters:

Task	Type	Loss/metric function
Regression	Loss/metrics	Mean squared error "mean_squared_error"
Classification	Loss/metrics	Cross entropy loss "categorical_crossentropy"

These metrics/loss functions are implemented using the keras.losses and keras.metrics modules.

The overall loss function for the whole model J is automatically defined in the compilation of the model.

Callbacks are also implemented to (1) save the best performing model based on minimum of J and (2) perform the early stopping in case overfitting occurs.

Decision tree metrics

The metrics for training and evaluating the decision tree model are the mean squared error and accuracy, in the DecisionTreeRegressor and DecisionTreeClassifier models respectively.

5. Experiment and results

For each neural network model and decision tree model, we do the following:
(1) Train the model and plot the model performance against varying dataset sizes.
(2) Select the dataset size giving the best model performance using our metrics.
(3) Train the model again to evaluate the model performance. In our neural network models, the model performance is measured against each epoch.

Neural networks

The training experiments are conducted as follows:

For each model n,

A subset of the dataset of size 10000 is sampled randomly.
The model is trained on that subset for E epochs, with batch size B.

Training metrics:
The MSE and Cross-entropy loss are recorded.
Validation metrics:
The MSE and Cross-entropy loss are recorded.

The model is evaluated on the test subset.

Test metrics:
The MSE and Cross-entropy loss are recorded.

Repeat the training and testing process with 10000 more samples, until It reaches 100000.

Each loss values are round to 3 d.p.

Here are the results:

Model 1:

sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	15.786	16.531	15.480	0	0	0
20000	14.956	14.980	15.104	0	0	0
30000	14.826	14.758	14.761	0	0	0
40000	14.551	15.109	14.296	0	0	0
50000	14.758	14.146	14.529	0	0	0
60000	14.657	14.703	14.682	0	0	0
70000	14.622	14.739	14.407	0	0	0
80000	14.652	14.653	14.402	0	0	0
90000	14.636	14.705	14.475	0	0	0
100000	14.619	14.693	14.501	0	0	0

Model 2:

sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	30.868	29.725	31.723	0	0	0
20000	6.061	6.077	5.892	0	0	0
30000	5.731	5.692	5.859	0	0	0
40000	5.943	5.706	5.904	0	0	0
50000	6.003	5.891	5.817	0	0	0
60000	5.695	5.826	5.638	0	0	0
70000	5.740	5.797	5.616	0	0	0
80000	5.576	5.561	5.535	0	0	0
90000	5.822	5.836	5.961	0	0	0
100000	5.703	5.565	5.724	0	0	0

Model 3:

sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	NaN	NaN	NaN	NaN	NaN	NaN
20000	NaN	NaN	NaN	NaN	NaN	NaN
30000	NaN	NaN	NaN	NaN	NaN	NaN
40000	NaN	NaN	NaN	NaN	NaN	NaN
50000	NaN	NaN	NaN	NaN	NaN	NaN
60000	NaN	NaN	NaN	NaN	NaN	NaN
70000	NaN	NaN	NaN	NaN	NaN	NaN
80000	NaN	NaN	NaN	NaN	NaN	NaN
90000	NaN	NaN	NaN	NaN	NaN	NaN
100000	NaN	NaN	NaN	NaN	NaN	NaN

Model 4:

Sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	14.776	15.393	14.739	0	0	0
20000	14.806	15.414	14.645	0	0	0
30000	14.721	14.463	14.263	0	0	0
40000	15.024	14.652	14.681	0	0	0
50000	14.831	14.422	14.571	0	0	0
60000	14.765	14.926	14.739	0	0	0
70000	14.945	15.138	14.784	0	0	0
80000	14.810	14.511	14.613	0	0	0
90000	15.421	15.276	15.101	0	0	0
100000	14.674	14.617	14.527	0	0	0

Model 5:

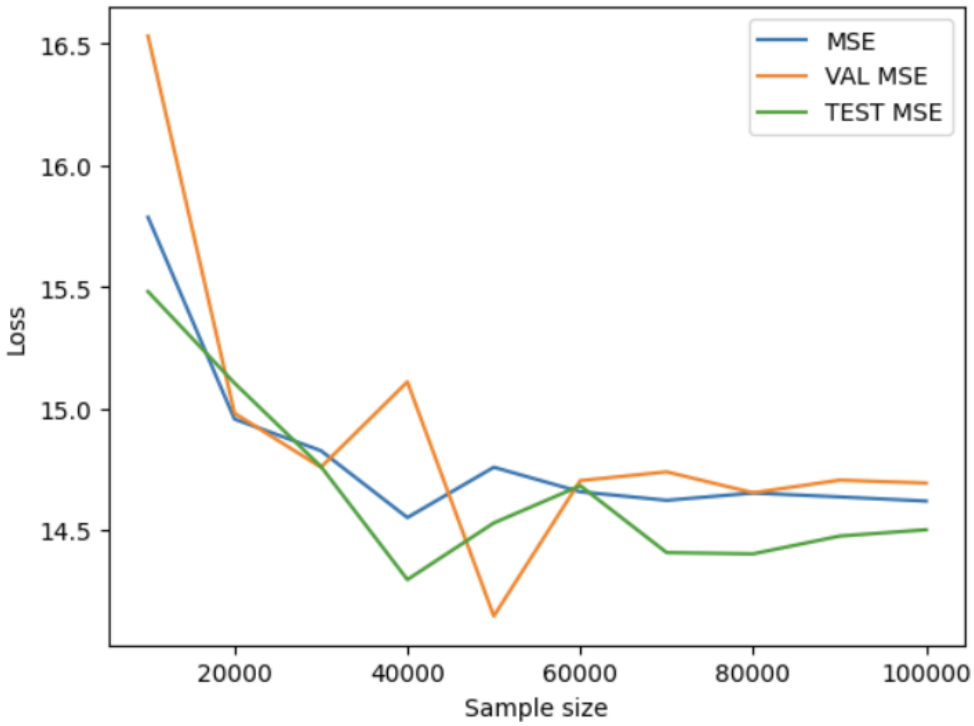
sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	5.750	6.264	5.796	0	0	0
20000	5.629	5.847	5.834	0	0	0
30000	5.551	5.583	5.605	0	0	0
40000	5.625	5.763	5.892	0	0	0
50000	5.629	5.857	5.805	0	0	0
60000	5.607	5.887	5.622	0	0	0
70000	5.624	5.624	5.534	0	0	0
80000	5.626	5.555	5.556	0	0	0
90000	5.652	5.632	5.659	0	0	0
100000	5.647	5.650	5.590	0	0	0

Model 6:

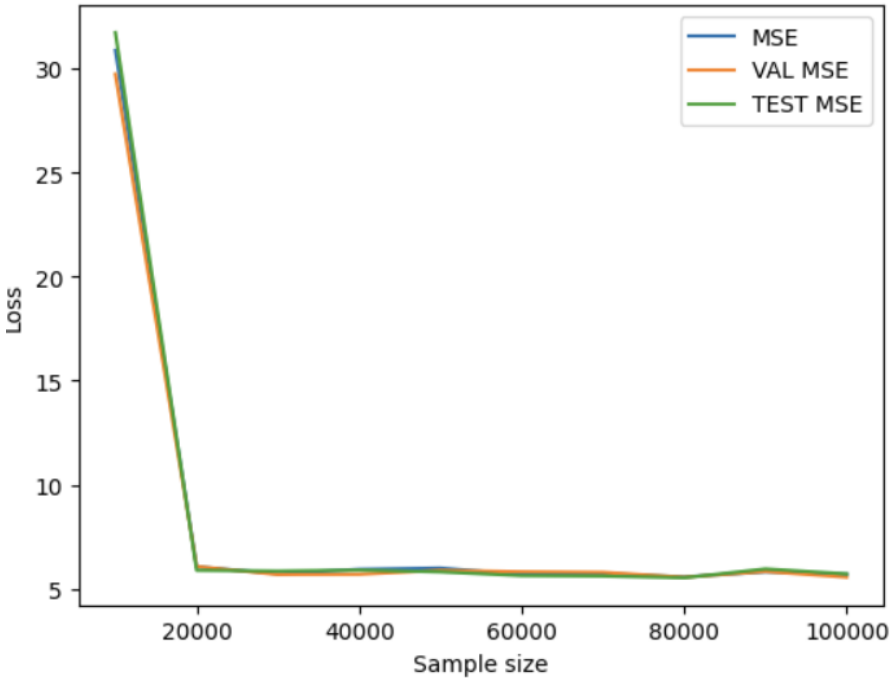
sample	MSE	val_MSE	Test_MSE	CEL	val_CEL	Test_CEL
10000	NaN	NaN	NaN	NaN	NaN	NaN
20000	NaN	NaN	NaN	NaN	NaN	NaN
30000	NaN	NaN	NaN	NaN	NaN	NaN
40000	NaN	NaN	NaN	NaN	NaN	NaN
50000	NaN	NaN	NaN	NaN	NaN	NaN
60000	NaN	NaN	NaN	NaN	NaN	NaN
70000	NaN	NaN	NaN	NaN	NaN	NaN
80000	NaN	NaN	NaN	NaN	NaN	NaN
90000	NaN	NaN	NaN	NaN	NaN	NaN
100000	NaN	NaN	NaN	NaN	NaN	NaN

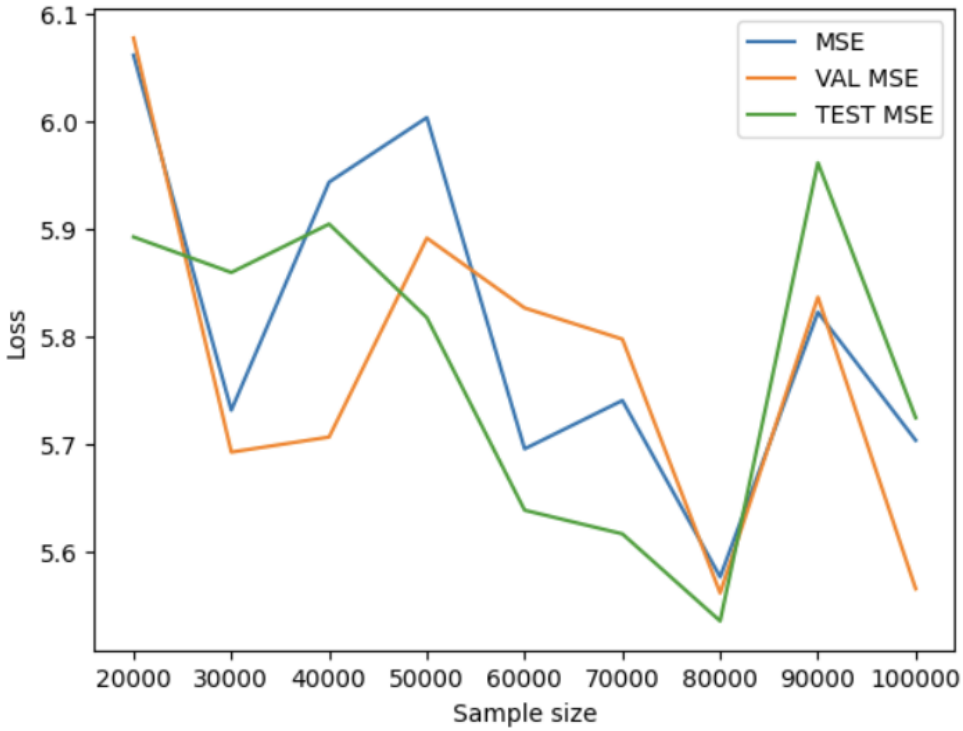
Visualizing the mean square losses:

Model 1:

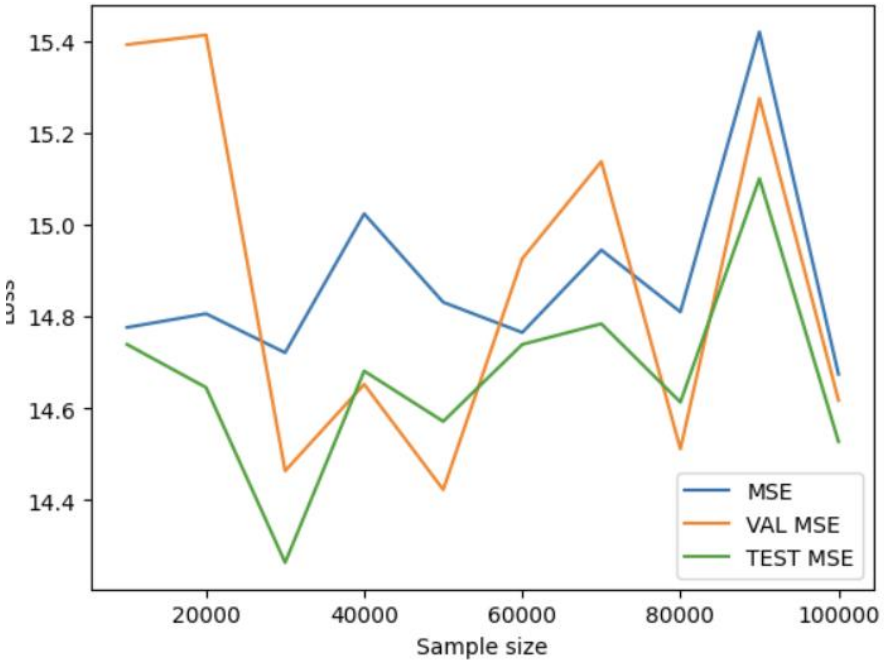


Model 2:

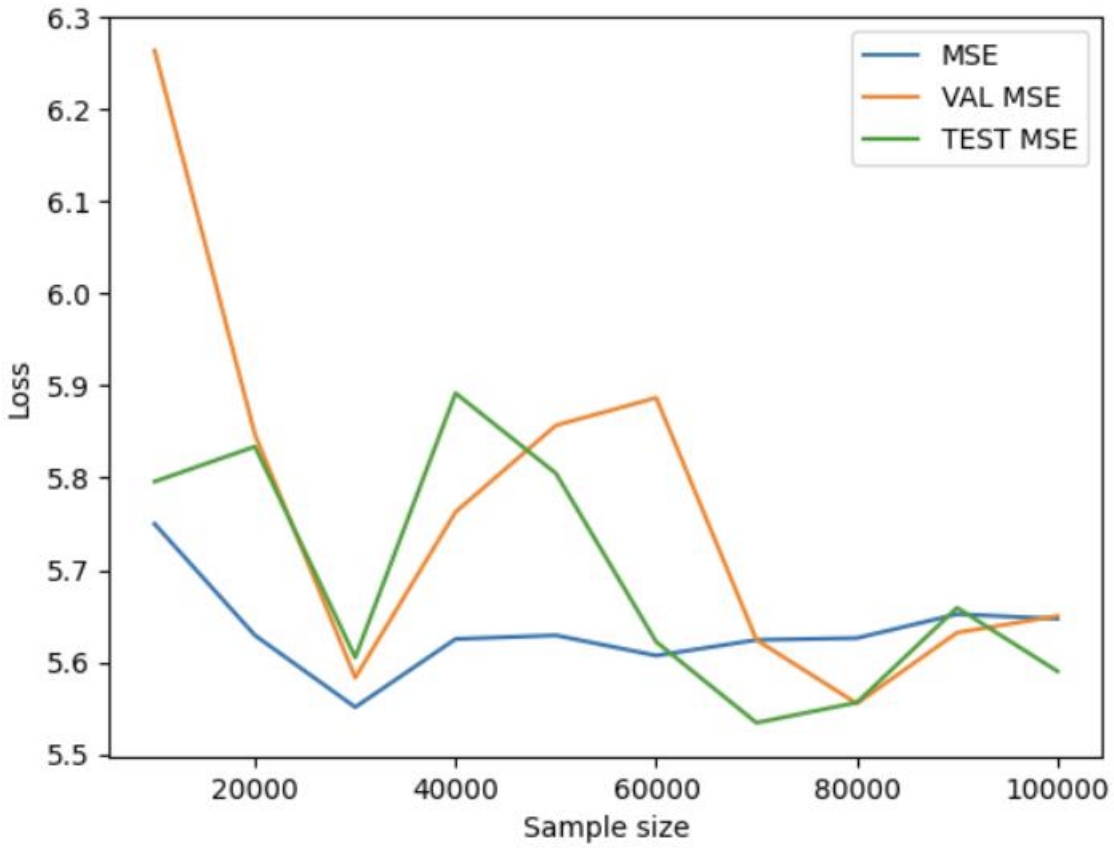



MSE
VAL MSE
TEST MSE
Loss
6.1
6.0
5.9
5.8
5.7
5.6
20000
30000
40000
50000
60000
70000
80000
90000
100000
Sample size

Model 4:

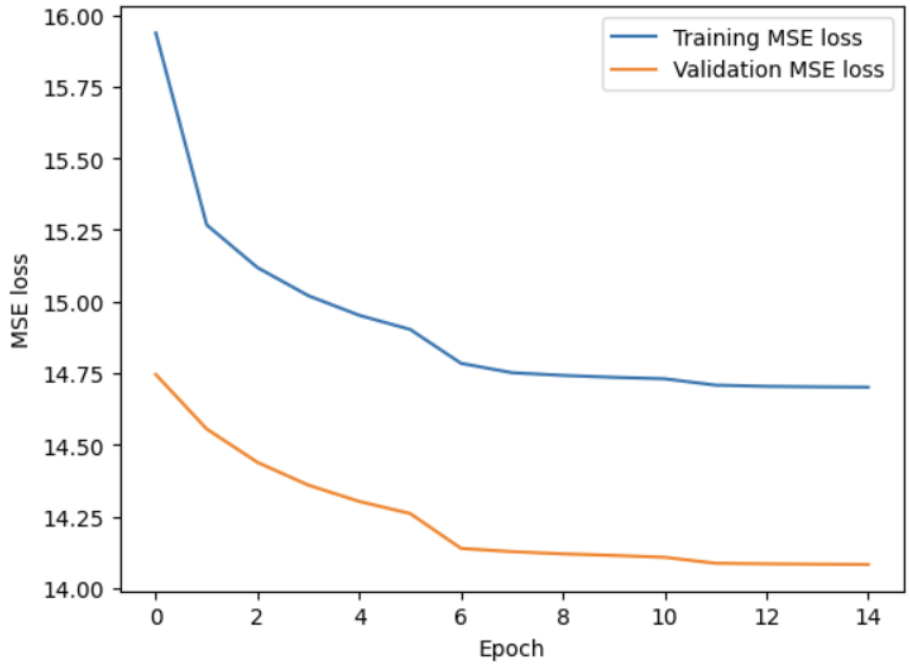

MSE
VAL MSE
TEST MSE
15.4
15.2
15.0
14.8
14.6
14.4
20000
40000
60000
80000
100000
Sample size

Model 5:



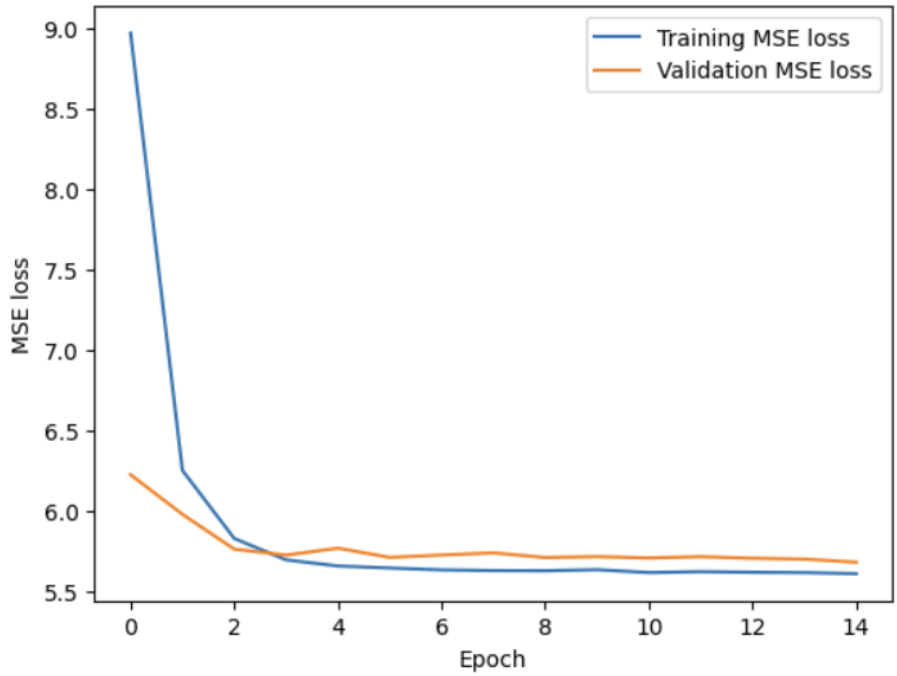
Train with the dataset size with best model performance again:

Model 1: 40000 samples



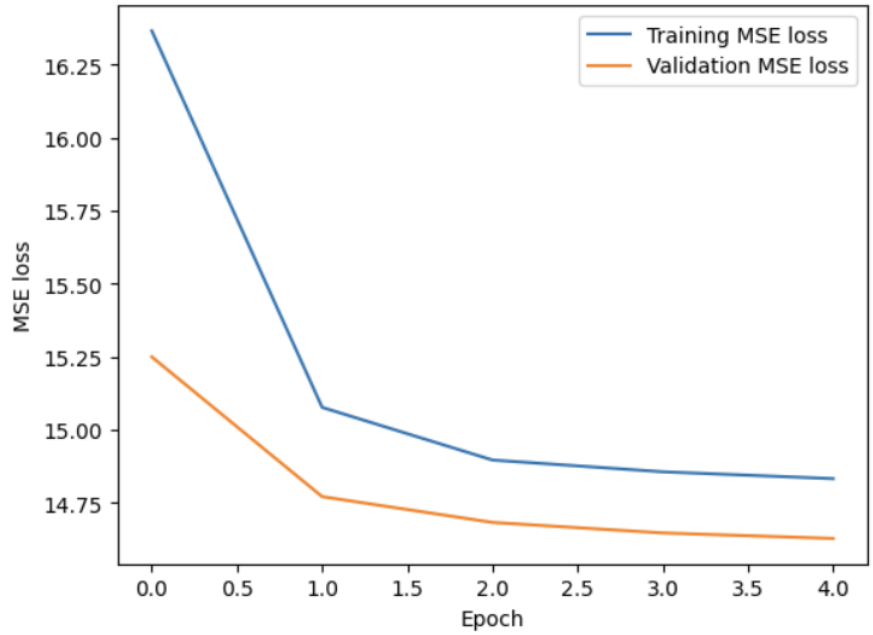
MSE on test set: 14.365

Model 2: 80000 samples



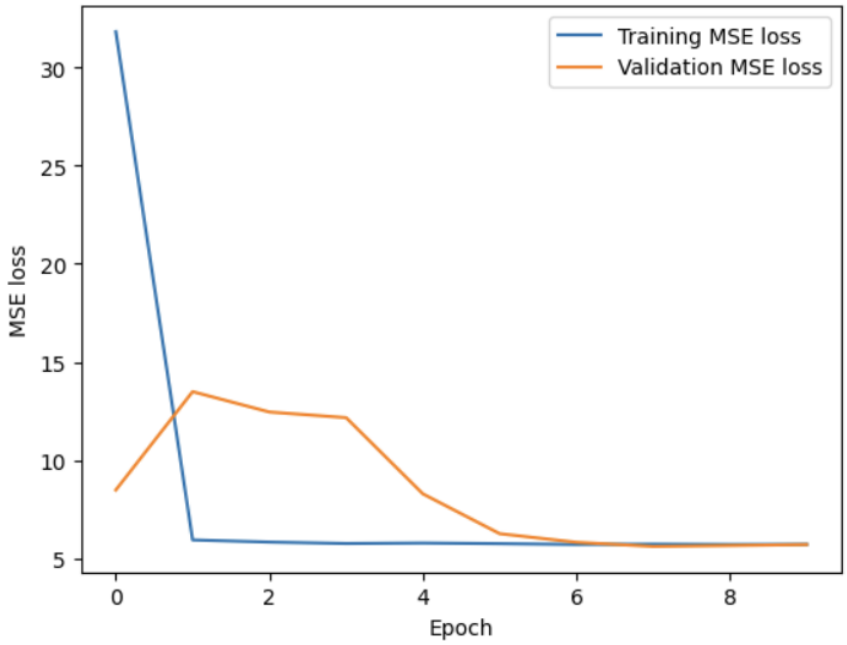
MSE on test set: 5.580

Model 4: 30000 samples



MSE on test set: 14.280

Model 5: 30000 samples



MSE on test set: 5.759

Decision trees:

The training experiment flow are as follows:

Select a random sample of size 1000 is extracted from the dataset.

Initialize the decision tree regressor and classifier skeletons.

The models are fitted with the training input features against the training labels.

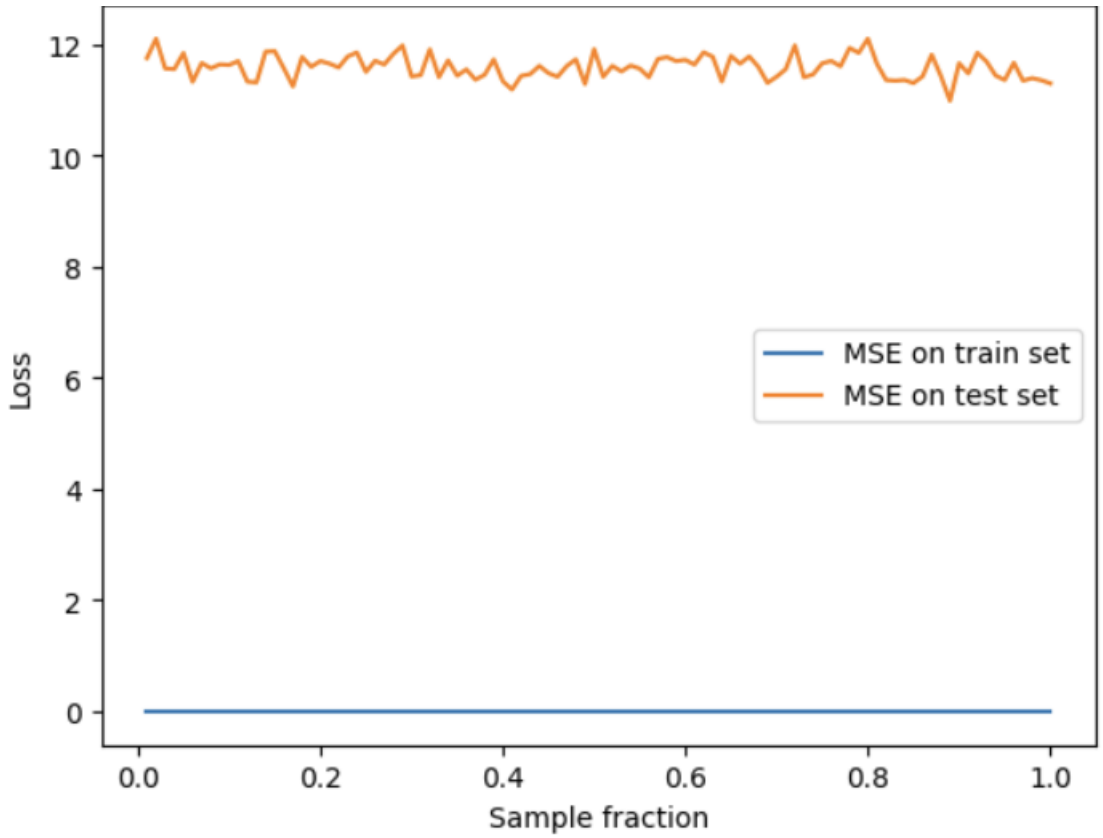
From the training dataset, the MSE score is calculated for regressor model, and the accuracy score is calculated for classifier model,

The predicted outputs are inferred from the testing input features.

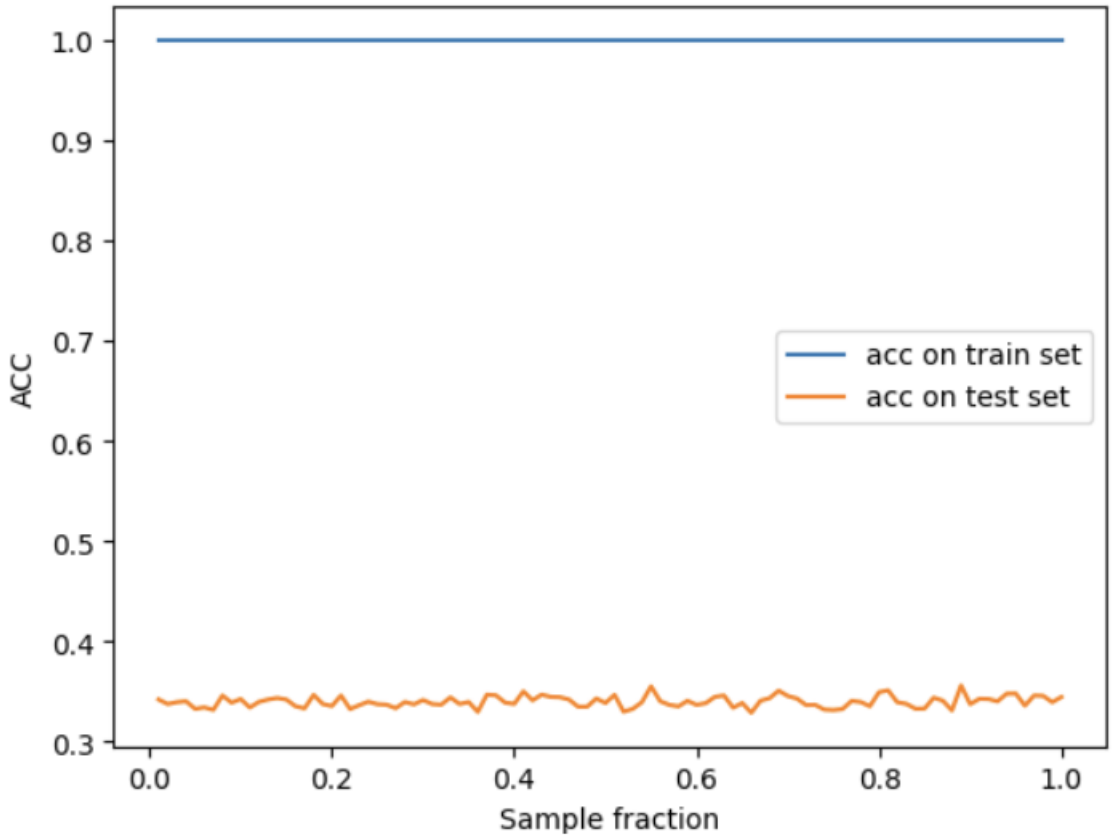
The MSE score is calculated for regressor model, and the accuracy score is calculated for classifier model,

Repeat for 1000 more samples until it reaches 100000 samples.

Here are the results:



Tree depth: 63



Tree depth: 67

Best decision trees and model performance score:
Dataset size: 49000
MSE: 11.631, ACC: 0.3485

6. Discussion

On neural network models

1. On the usage of autoencoder
   Notice that the MSE of models without using autoencoder (Model 1, 2, 3) is about the same as that of models using the autoencoder (4, 5, 6).
   Autoencoders have the purpose of performing dimensionality reduction and denoising. Specifically, the encoder transforms the data in its original dimension to its smaller dimensions, by filtering off noisy data, irrelevant input features or purely compressing the data.
   However, the autoencoder used in our training does not contribute to a better model performance, since its sole purpose is to compress or denoise any data.
   From this observation, there might be a threshold of size of bottleneck layer $\boldsymbol{F}$. Widening the bottleneck layer up to $\boldsymbol{F}$ lowers the training loss, and further widening may not cause significant decrease in training loss. This is because in inputs for classification/regression models in general, we only require more relevant input attributes useful to these models. That might be the same principle of feature selection, where we select enough input attributes for optimal performance, while preventing underfitting from overly few input attributes.

2. On the usage of ResNet

   Suppose we do not use the Common ResNet at all. Then we train model 5, the best model we recorded. Then we observe the following:

	MSE_loss	Val_MSE_loss	Test_MSE_loss
With ResNet	5.551	5.583	5.605
Without ResNet	5.635	5.782	5.569

   As we can see, the Common ResNet contributes to a slight improvement in model performance, but the difference of the validation loss is slightly larger than that of training and test losses. It has nothing to do with the ResNet structure, but rather the fully connected layers contributing to the reduction of underfitting.

   However, if we remove the skip connection and trained model 5 as shown below,

	MSE_loss	Val_MSE_loss	Test_MSE_loss
With ResNet	5.551	5.583	5.605
No ResNet	5.635	5.782	5.569
No skip connect	5.675	5.636	5.604

   there will be an overall increase in losses, thanks to lower retention of original data and neuron gradients. Nevertheless, the change of losses is still small, as our ResNet design is relatively shallow; and such loss change will theoretically increase over deeper networks.

3. On choice of activation functions

   In our training process, we found that activation functions may have caused the greatest impact on model performance.

   As seen from the table below

Model	Activation function	MSE_loss	Val_mse_loss	Test_mse_loss
1	Sigmoid	14.551	15.109	14.296
4	Sigmoid	14.721	14.463	14.263
2	ELU	5.576	5.561	5.535
5	ELU	5.551	5.583	5.605
3	ReLU	NaN	NaN	NaN
6	ReLU	NaN	NaN	NaN

   the losses of models using ELU(Exponential Linear Unit) is significantly lower than that of models using sigmoid functions,

but ReLU functions instead triggers either an exploding or vanishing gradient problem as seen from losses reaching $-\infty$ or ∞.

The input features are normalized prior to training, where they follow a standard normal distribution.

Theoretically, the output range of a sigmoid function is restricted to $(0,1)$. For our input features however, that range is even smaller, $[a, b]$, possibly where $0 < a < b < 1$. The output range may even be a singleton $\{a\}$ in the worst-case scenario. As we can see, the range of intermediate values of a layer might be smaller than that of the previous layer, and that range is likely to shrink with each layer deeper, that range is unlikely to include 0. As a result, the neural gradients of the next layer are more likely to be smaller than that of the previous layer, so the gradients are indeed vanishing and likely to be a cause for a significantly high MSE loss.

The output range of the ELU function, however, is much wider than that of the sigmoid, at $(-1, \infty)$. Even with each deeper layer, the possible output range is still $(-1, \infty)$ given our input features. This increases the rate of retention of intermediate values, especially when negative values are taken.

The ReLU function, unfortunately, is likely the main cause of failure of training Models 3 and 6. It is because we have a sizeable number of negative values in our whole batch of normalized input attributes. The ReLU functions has an output range of $[0, \infty)$. Albeit light on computational resources, all negative input values will instantly outputted as 0 and affected gradients would stop backpropagation learning to zero partial derivative.

Therefore, the optimal choice for activation functions would be "elu".

4. On the SLP classifier loss
   Interestingly, the accuracy of our classification layer component is consistently 0. This means that there is a 100% chance of labelling the right class, given the set of input attributes, whether it is for training or testing. It is because the number of parameters on our

neural network is far more than needed to build a model giving optimal classification performance.

On decision tree models

The decision tree regressor has attained a performance score between the regressor using ELU and the regressor using sigmoid function.

5. On the low classification accuracy on the test dataset

   Compared to the classification module in our neural network model, our decision tree classifier suffers from a low accuracy score.

   This may be due to the large size of input attributes, plus they are normalized and therefore discontinuous. The discontinuity of the values of input attributes increases the difficulty of model generalization and significantly deepens the decision tree. When the tree is being built, it is consequently trained to tackle the specificities of the discontinuous training data, but instead lowers the generality in relation to non-training data.

   To improve the accuracy of the classifier, we need to perform two more preprocessing steps on the dataset:

   (1) Normalization of all input attributes should be skipped.
   (2) PCA of dimension 3 should be performed on the input dataset.
   (3) Map any non-discrete values to discrete values by floor or ceiling functions.

   Another solution is to replace singular decision tree models with random forest models to improve accuracy. This is done by generating sets of smaller decision tree models trained on random splits of input features, and the performance would be boosted.

7. Conclusion

It is utmost priority to accurately predict the possible length of stay and readmission count to efficiently manage hospital resources.

By using the given dataset to train our neural network and decision tree models, we discovered and analyzed the ups and downs of our neural network models and its components, and the decision tree model.

Even though our decision tree models are straightforward to implement, neural network models offer optimal performance in its regression and classification applications.