A Comparison of Statistical Models and Deep Learning for Data with Binary Response and Longitudinal Covariates

by
Zhikang Chen

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Mathematics Statistics + Artificial Intelligence.

Guelph, Ontario, Canada

ABSTRACT

A COMPARISON OF STATISTICAL MODELS AND DEEP LEARNING FOR DATA WITH BINARY RESPONSE AND LONGITUDINAL COVARIATES

Zhikang Chen
University of Guelph, 2021

Advisor:
Dr. Julie Horrocks

In statistics, longitudinal data refers to data in which the response variable and explanatory variables are measured several times for each subject. However, in the machine learning literature, longitudinal data can also refer to data in which only the explanatory variables are repeatedly measured, but not the response variable. This thesis compared two statistical models - the baseline logistic regression and the two-stage joint model, and two neural network approaches - the feed-forward neural network and the recurrent neural network with long short-term memory, in terms of the prediction sensitivity, specificity, area under the receiver operating characteristic curve, and Brier score. Data analysis was conducted using data from two clinical trials and a simulation study was also conducted. For the datasets generated and studied in this thesis, the neural network approaches show no advantages compared to the other statistical methods.

Dedication

To my wife, my parents, and my parent-in-laws, who encouraged, supported, and loved me throughout my education and life.

Acknowledgements

I would like to thank my advisor, Dr. Julie Horrocks, for all her advice, support, encouragement, trust and patience throughout the past years, especially during this pandemic.

I would also like to thank my committee member, Dr. Gerarda Darlington, for taking the time out of her busy schedule to provide valuable opinion and feedback on my thesis.

Finally, I would like to show my appreciation to the research scientist at McMaster University, Dr. Weiguang Guan, and my deparment friends Alysha Cooper, Matthew Lowe and Glen Reavie, for their help resolving technical issues within SHARCNET.

Contents

List of Tables

Chapter 1

Introduction

The term "longitudinal data" represents a type of data that records the information from the same subject multiple times. Sometimes, longitudinal data is also referred as "panel data" (Blundell and Mátyás, 1992). In contrast, cross-sectional data measures each subject only once. Statistical methods for longitudinal data, including the generalized linear mixed model (GLMM; Breslow et al., 1993) and generalized estimating equations (GEE; Zeger et al., 1988), were developed to fit data that has both explanatory and response variables repeatedly measured.

However, in the machine learning literature, longitudinal data can also refer to data in which only the explanatory variables are repeatedly measured, but not the response variable. Instead, the response variable can be a binary or time-to-event outcome, as in Che et al. (2017), Choi et al. (2017), Zhang et al. (2018), Reddy and Delen (2018), Falissard et al. (2018).

This thesis will focus on comparing four methods for modelling data with binary response and repeatedly measured longitudinal covariates. Two of them are statistical models: logistic regression (Hosmer et al., 2013) and joint two-stage model (Horrocks and van Den Heuvel, 2009), and the other two are popular deep learning approaches: feed-forward neural network

(Zhang et al., 2020) and recurrent neural network (Elman, 1990). The statistical software `R` (R Core Team, 2020) as well as several packages are used as computational tools for model fitting and evaluation. The `R` package `keras` (Allaire and Chollet, 2020) and `tensorflow` (Allaire and Tang, 2020) are used for training the neural networks. Two datasets will be analyzed: the `liver` dataset from the package `joineR` (Philipson et al., 2018) and the `pbc2` dataset from the package `JM` (Philipson et al., 2018). Recorded during cirrhosis treatment trials, both datasets have various longitudinal covariates and a response variable, the time of death or censoring. However, this thesis will consider a different response variable – whether the subject survived for a fixed period of time, which is a binary classification problem. During the model fitting and evaluation, both datasets will be split into training and test sets at a ratio of 70% to 30%.

To compare the models, this thesis will use evaluation metrics calculated from the test set, such as the Area Under the Receiver Operating Characteristic Curve (AUC; Fawcett, 2006), true positive rate (or sensitivity; Powers, 2011) and true negative rate (or specificity; Powers, 2011), and Brier score (BS; Brier, 1950).

The organization of the thesis is as follows: the logistic regression model, two-stage model and the neural networks will be introduced in Chapter 2. The model evaluation methods which include sensitivity, specificity, receiver operating characteristic (ROC) curve, the Brier Score (BS), will be introduced in Chapter 3. The model fitting and result comparisons for both datasets will be presented in the Chapter 4. A simulation study will then be conducted and discussed in Chapter 5, followed by Chapter 6 which will bring up ethical issues related to the neural networks used in this thesis. Lastly, Chapter 7 will conclude the thesis and suggest possible future work. All the computation source code for statistical software `R` (R Core Team, 2020) and SHARCNET (Compute Canada) task submission is attached at the end of the thesis, see Appendix A.

Chapter 2

Methods for Longitudinal Data Classification

This thesis will focus on four classification methods: 1) the conventional logistic regression model for binary response variable (Tolles et al., 2016), 2) a two-stage joint model (Horrocks and van Den Heuvel, 2009), and the neural networks approach which includes 3) feed-forward neural network (Biganzoli et al., 1998) and 4) recurrent neural network (Elman, 1990).

The conventional logistic regression does not support longitudinal covariates. A common work-around is to convert the longitudinal covariates into cross-sectional covariates by dropping time-related information. In this thesis, I will fit the logistic regression model using the information available at the first time point for each subject. However, doing so means the loss of a significant amount of information in the data.

The joint model for binary and longitudinal data (Horrocks and van Den Heuvel, 2009; Wang et al., 2000) is a method which can support a single longitudinal covariate. The first stage is a linear mixed effects model fit to the longitudinal covariate, and the second stage is a logistic regression that uses the estimated random effects as covariates along with time-fixed covariates.

Neural networks can conduct classification on a binary response variable. While the most widespread models are the feed-forward neural networks, the recurrent neural networks work well especially for time-series or longitudinal data (Goodfellow et al., 2016). Neural networks approximate some linear or non-linear function that maps the covariates to the response variables, and seek the parameters that provide the best approximation. One difference between the feed-forward and recurrent neural networks is that for the feed-forward neural networks, information is passed only forward through the layers without flowing back, and for the recurrent neural network, the information is passed both forward and backward. Moreover, there is a hidden state in the recurrent neural network which incorporates information from the past timesteps.

To use the feed-forward neural network with longitudinal covariates, the longitudinal information needs to be converted to either cross-sectional information, or summarized information, which will entail information loss. Furthermore, if the number of measurements varies by subject, the length of the covariate vectors will also vary, which will prevent the use of a feed-forward neural network. A simple solution, used in this thesis, is to conduct classification using the feed-forward neural networks with information available at the first time for each subject. The recurrent neural network overcomes the above problems and furthermore, is able to exploit the correlation between the longitudinal measurements from the same subject, as both previous and current information is used at each iteration. This thesis will conduct classification using the recurrent neural network with complete data.

2.1 Logistic Regression

Logistic regression is one of the most widely used models to fit data with a binary response variable. Assume that the response variable y_i is a binary outcome which indicates whether

an event occurs for the i-th subject, $i = 1, 2, ..., N$:

$$y_i = \begin{cases} 1, & \text{if an event occurs for the } i\text{-th subject} \\ 0, & \text{otherwise.} \end{cases}$$

The multiple logistic regression model could then be written as:

$$\log\left(\frac{p_i}{1-p_i}\right) = \mathbf{x}_i^{'}\boldsymbol{\beta},$$

where p_i is the probability of the event for the i-th subject, $\mathbf{x}_i$ is the covariate vector of dimension $p \times 1$ for this subject, and $\boldsymbol{\beta}$ is the $p \times 1$ vector of regression coefficients. Note that the binary outcome, y_i, is assumed to follow a Bernoulli distribution with parameter p_i.

In `R`, a built-in function called `glm` (R Core Team, 2020) can be used to fit a logistic regression model to the data by setting the argument `family='binomial'`.

2.2 Joint Model

Another approach for binary outcomes classification is the joint modelling framework consisting of a linear mixed effects submodel for the longitudinal covariates and generalized linear submodel for the binary outcomes as in Wang et al. (2000), and Horrocks and van Den Heuvel (2009). Assume that $\boldsymbol{W}_i = (W_{i1}, W_{i2}, ..., W_{im_i})$ is the vector of longitudinal measurements for the i-th subject, $i = 1, 2, ..., N$, at times $t_{i1}, t_{i2}, ..., t_{im_i}$. The linear mixed effects submodel for the longitudinal covariates can then be written as

$$\boldsymbol{W_i} = \boldsymbol{X_i}\boldsymbol{\alpha} + \boldsymbol{Z}_i\boldsymbol{U}_i + \boldsymbol{\epsilon}_i$$

where $\boldsymbol{X}_i$ and $\boldsymbol{Z}_i$ are design matrices of $m_i \times p$ and $m_i \times q$, respectively, $\boldsymbol{\alpha}$ is a fixed-effects parameter vector of dimension $p \times 1$, and $\boldsymbol{U}_i$ is a random-effects parameter vector of dimension $q \times 1$, and $\boldsymbol{\epsilon}_i = (\epsilon_{i1}, \epsilon_{i2}, ..., \epsilon_{im_i})$ is a measurement error vector of dimension $m_i \times 1$ for the i-th subject. It is assumed that

$$\boldsymbol{U}_i \overset{i.i.d.}{\sim} MVN(\boldsymbol{0}, \boldsymbol{\Sigma})$$

and

$$\boldsymbol{\epsilon}_i \overset{i.i.d.}{\sim} MVN(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_{m_i})$$

where vectors $\boldsymbol{U}_i$ and $\boldsymbol{\epsilon}_i$ are independent, $\boldsymbol{\Sigma}$ is a $q \times q$ covariance matrix for the random effects $\boldsymbol{U}_i$, and $\boldsymbol{I}_{mi}$ is the $m_i \times m_i$ identity matrix. Let y_i denote a binary outcome which indicates whether an event occurs for the i-th subject:

$$y_i = \begin{cases} 1, & \text{if an event occurs for the } i\text{-th subject} \\ 0, & \text{otherwise.} \end{cases}$$

The generalized linear submodel with logit link can be written as

$$\log\left(\frac{p_i}{1-p_i}\right) = \boldsymbol{V}_i\boldsymbol{\beta} + \boldsymbol{U}_i\boldsymbol{\gamma},$$

where p_i is the probability that $y_i = 1$, $\boldsymbol{V}_i$ is a vector of covariates that have fixed effects $\boldsymbol{\beta}$, thought to affect the binary response for the subject i, and $\boldsymbol{\gamma}$ is a vector of fixed effects associated with the random effects $\boldsymbol{U}_i$. The dimensions of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are $m_i \times 1$ and $q \times 1$, respectively. Horrocks and van Den Heuvel (2009) used Bayesian methods to estimate the parameters of the joint model.

In practice, a two-stage approach can be used to fit the joint model and make predictions.

The first stage is to fit a linear mixed model to the longitudinal data, $\boldsymbol{W}_i$, to obtain the best linear unbiased prediction of the random effects, $\hat{\boldsymbol{U}}_{\boldsymbol{i}}$ (Wang et al., 2000). A logistic regression model can then be fit in the second stage, by using the fitted random effects, $\hat{\boldsymbol{U}}_{\boldsymbol{i}}$, as covariates. Note that the two-stage approach can lead to biased estimates and standard errors, as the error in random coefficients is ignored at the second stage (Wang et al., 2000). However, it is easy to fit with available software, and will be used in this thesis. In `R` (R Core Team, 2020), the built-in function `glm` and function `lme` from package `nlme` (Pinheiro et al., 2020) can be used to fit the first and second stages of the joint model, respectively.

When performing prediction of survival the test set, we need a prediction of $\boldsymbol{U}_{\boldsymbol{i}}$ for individuals in the test set. Let superscript * indicate the value of a variable or random effect for an individual in the test set. We need to predict $\boldsymbol{U_i}^*$ given $\boldsymbol{W_i^*}$, $\boldsymbol{X_i^*}$, and $\boldsymbol{V_i^*}$, using the estimated parameters of the model fit to the training set: $\hat{\boldsymbol{\alpha}}$, $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\gamma}}$. As far as I know, this problem has not been studied in the literature, so I used the following procedure proposed by my supervisor. First, the residual, $\boldsymbol{e_i^*}$, of the first-stage response variable, $\boldsymbol{W}_i^*$,was calculated as:

$$\boldsymbol{e}_i^* = \boldsymbol{W}_i^* - \boldsymbol{X}_i^* \hat{\boldsymbol{\alpha}},$$

where $\boldsymbol{W_i^*}$ is a response variable vector for subject i in the test set, $\boldsymbol{X_i^*}$ is a $m_i \times p$ design matrix for the test set, $\hat{\boldsymbol{\alpha}}$ is the fitted parameter vector for the training set. Then a linear mixed effects model was fit to the residual:

$$\boldsymbol{e}_i^* = \boldsymbol{Z_i}^* \boldsymbol{U_i}^* + \boldsymbol{\epsilon_i}^*,$$

where $\boldsymbol{Z_i^*}$ is a $m_i \times q$ design matrix for the test set, $\boldsymbol{U_i^*}$ is an unknown random-effects parameter vector to be estimated of dimension $q \times 1$, and $\boldsymbol{\epsilon}_i^* = (\epsilon_{i1}^*, \epsilon_{i2}^*, ..., \epsilon_{im_i}^*)$ is an unknown measurement error vector of dimension $m_i \times 1$ for the i-th subject for the test set, $i =$

$1, 2, ..., N^*$, at times $t_{i1}, t_{i2}, ..., t_{im_i}$. It is assumed that

$$\boldsymbol{U_i^*} \sim N(\mathbf{0}, \boldsymbol{\Sigma}^*)$$

and

$$\boldsymbol{\epsilon_i^*} \sim N(\mathbf{0}, \sigma^{*2} \boldsymbol{I}_{m_i}),$$

where vectors $\boldsymbol{U_i^*}$ and $\boldsymbol{\epsilon_i^*}$ are independent, $\boldsymbol{\Sigma^*}$ is a $q \times q$ covariance matrix for the random effects $\boldsymbol{U_i^*}$, and $\boldsymbol{I}_{m_i}$ is the $m_i \times m_i$ identity matrix.

The best linear unbiased estimation of the random effect, $\hat{\boldsymbol{U_i}}^*$, will be used for the second-stage prediction, to estimate the response variable p_i^*. Specifically:

$$\hat{p}_i^* = \exp(\boldsymbol{V}^*_i \hat{\boldsymbol{\beta}} + \hat{\boldsymbol{U}}_{\boldsymbol{i}}^* \hat{\boldsymbol{\gamma}}) / (1 + \exp(\boldsymbol{V}^*_i \hat{\boldsymbol{\beta}} + \hat{\boldsymbol{U}}_{\boldsymbol{i}}^* \hat{\boldsymbol{\gamma}})),$$

where $\boldsymbol{V_i^*}$ is a fixed effects vector of covariates that affect the binary outcome for test set subject i, $i = 1, 2, ..., N^*$, $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\gamma}}$ are the estimated parameter vectors for the training set.

2.3 Neural Networks

An artificial neural network (ANN) is a set of dynamic systems of linear and non-linear equations that are capable of modelling nonlinear processes. Although neural networks have been mostly developed by computer scientists, engineers and neurophysiologists, many of them are reinventions of known statistical or mathematical methods (Zhang et al., 2020). Terminology in the neural networks literature is different from the statistical literature. For example, response variables are called target variables, and explanatory variables are called features (Zhang et al., 2020).

Deep neural networks, one type of ANN, consist of an input layer, an output layer, and

multiple hidden layers in between. Each layer includes nodes, which perform linear or non-linear calculations based on the layer's activation function and parameters. Consecutive layers are bridged by connections between their respective nodes. There are usually multiple inputs and outputs for each node.

In general, a neural network training process is composed of two iterative processes: forward propagation and backpropagation (Zhang et al., 2020). In forward propagation, the neural network is fed with the training data and computations occur through all layers from the input layer to the output layer to make predictions on the response variable. In backpropagation, the neural network calculates the gradient of the loss function with respect to the weights (see Section 2.3.3) to update the parameters in each node. These two steps are performed iteratively until some stop criteria are satisfied. The `R` (R Core Team, 2020) package `keras` (Allaire and Chollet, 2020) provides a high-level neural network programming interface similar to the widely used `Python` (Python Core Team, 2020) library `Keras` (Chollet et al., 2015) .

2.3.1 The Feed-forward Neural Network

For a dataset with N observations, assume that $\boldsymbol{x}_i^o = [x_{i1}^o, x_{i2}^o, ..., x_{iJ}^o]'$ are the observed J covariates and $\boldsymbol{y}_i^o = [y_{i1}^o, y_{i2}^o, ..., y_{iK}^o]'$ are the observed K response variables for subject i, $i = 1, 2, ..., N$. Note that for this thesis, $K = 1$. For an L-layer feed-forward neural network, the first layer and the L-th layer are the input and output layer, respectively, and the other $L - 2$ layers are hidden layers. At the l-th layer (Biganzoli et al., 1998):

$$\boldsymbol{s_i^{l+1}} = g_l(\boldsymbol{w}_l \boldsymbol{s_i^l} + \boldsymbol{\beta}_l),\ l = 1, 2, ..., L - 1, \tag{2.1}$$

where $\hat{y}_i = \boldsymbol{s_i^L}$ is the predicted value of response variables, the observed response variables, $\boldsymbol{y_i^o}$, are used for loss calculation during the training process, $\boldsymbol{x}_i^o = \boldsymbol{s_i^1}$ is the observed covariate

vector for subject i, respectively, $\boldsymbol{w}_l$ and $\boldsymbol{\beta}_l$ are the weight matrices and bias vectors at the l-th layer, respectively, and g_l is the activation function at the l-th layer. Note that the dimensions of $\boldsymbol{w}_l$ and $\boldsymbol{\beta}_l$ are $\alpha_{l+1} \times \alpha_l$ and $\alpha_{l+1} \times 1$, respectively, where α_l represents the number of nodes in the l-th layer, $\alpha_1 = J$ and $\alpha_{L-1} = K$. Equation 2.1 can be written as:

$$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = g_{L-1}(\boldsymbol{w}_{L-1} g_{L-2}(\cdots g_2(\boldsymbol{w}_2 g_1(\boldsymbol{w}_1 \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{o}} + \boldsymbol{\beta}_1) + \boldsymbol{\beta}_2) \cdots) + \boldsymbol{\beta}_{L-1}),$$

which is commonly referred as "forward propagation" (Zhang et al., 2020). In `R` (R Core Team, 2020), the package `keras` (Chollet et al., 2015) can be used to construct a feed-forward layer by using the pipeline '`layer_dense()`'.

2.3.2 The Recurrent Neural Network

Consider a longitudinal dataset with N subjects and T timesteps, assume that $\boldsymbol{x_{i,t}}^o = [x^o_{i1,t}, x^o_{i2,t}, ..., x^o_{iJ,t}]'$ are the observed J covariates for subject i at timestep t, $i = 1, 2, ..., N$ and $t = 1, 2, ..., T$, and $\boldsymbol{y}_i^o = [y^o_{i1}, y^o_{i2}, ..., y^o_{iK}]'$ are the observed K response variables for subject i at the last timestep, T, $i = 1, 2, ..., N$. Note that in this thesis, $K = 1$. For a 3-layer recurrent neural network (Elman, 1990), with input, output and 1 hidden layer, at each timestep t:

$$\begin{gathered} \textit{hidden state: } \boldsymbol{s}_{i,t} = g_s(\boldsymbol{w}_s \boldsymbol{s}_{i,t-1} + \boldsymbol{u}_s \boldsymbol{x_{i,t}} + \boldsymbol{\beta_s}), \\ \boldsymbol{y_{i,t}} = g_y(\boldsymbol{w}_y \boldsymbol{s_{i,t}} + \boldsymbol{\beta}_y),\ i = 1, 2, ..., N;\ t = 1, 2, ..., T, \end{gathered} \tag{2.2}$$

where the predicted value of the response variable is $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{y_{i,T}}$, the observed response variables, $\boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{o}}$, are used for loss calculation during the training process, and the observed values of covariates fed to the neural network at timestep t are $\boldsymbol{x_{i,t}} = \boldsymbol{x_{i,t}}^o$, $i = 1, 2, ..., N$ and $t = 1, 2, ..., T$. The initial value of the hidden state, $\boldsymbol{s_{i,0}} = \vec{\boldsymbol{0}}$. The matrices $\boldsymbol{w}_s$, $\boldsymbol{u}_s$

and $\boldsymbol{w}_y$ are the weight matrices, $\boldsymbol{\beta_s}$ and $\boldsymbol{\beta_y}$, g_s and g_y are the bias vectors and activation function for the state s and the response variable y respectively. Assuming that there are α nodes in the hidden layer, the dimension of $\boldsymbol{w}_s$, $\boldsymbol{u}_s$ and $\boldsymbol{w}_y$ are $\alpha \times \alpha$, $\alpha \times J$ and $K \times \alpha$, repectively. The dimension of $\boldsymbol{\beta_s}$ and $\boldsymbol{\beta_y}$ are $\alpha \times 1$ and $K \times 1$, respectively. This model is called "many-to-one" recurrent neural network (Cheng, et al., 2020). In `R` (R Core Team, 2020), the package `keras` (Chollet et al., 2015) can be used to add a recurrent layer by using the pipeline `'layer_simple_rnn()'`.

Long Short-Term Memory

Equation 2.2 shows a standard recurrent neural network with a hidden state vector $\boldsymbol{s}$ that updates itself based on the observation of the covariates at timestep t, $\boldsymbol{x_{i,t}}$, and the hidden state vectors $\boldsymbol{s}$ from the most recent timestep $\boldsymbol{s_{i,t-1}}$. While this model can exploit the correlations between the observations within small timestep lags, it normally fails for large timestep lags due to either backpropagation error exploding or vanishing. In order to solve this problem, a special recurrent neural network architecture was developed - the long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997).

A LSTM unit features a forget gate, an input gate, an output gate, a memory cell, a cell state, and a hidden state, where the cell state carries the information from the previous gates and cell state, and the gates regulate the processing of the information for the cell state. The forward propagation process of a LSTM at timestep t is (Hochreiter and Schmidhuber, 1997):

$$\textit{forget gate: } \boldsymbol{f}_{i,t} = g_f(\boldsymbol{w}_f \boldsymbol{s}_{i,t-1} + \boldsymbol{u}_f \boldsymbol{x}_{i,t} + \boldsymbol{\beta}_f),$$

$$\textit{input gate: } \boldsymbol{I}_{i,t} = g_I(\boldsymbol{w}_I \boldsymbol{s}_{i,t-1} + \boldsymbol{u}_I \boldsymbol{x}_{i,t} + \boldsymbol{\beta}_I),$$

$$\textit{output gate: } \boldsymbol{o}_{i,t} = g_o(\boldsymbol{w}_o \boldsymbol{s}_{i,t-1} + \boldsymbol{u}_o \boldsymbol{x}_{i,t} + \boldsymbol{\beta}_o),$$

$$\textit{memory cell: } \boldsymbol{m}_{i,t} = g_m(\boldsymbol{w}_m \boldsymbol{s}_{i,t-1} + \boldsymbol{u}_m \boldsymbol{x}_{i,t} + \boldsymbol{\beta}_m),$$

$$\textit{cell state: } \boldsymbol{c}_{i,t} = \boldsymbol{f}_{i,t} \circ \boldsymbol{c}_{i,t-1} + \boldsymbol{I}_{i,t} \circ \boldsymbol{m}_{i,t},$$

$$\textit{hidden state: } \boldsymbol{s}_{i,t} = \boldsymbol{o}_{i,t} \circ g_s(\boldsymbol{c}_{i,t}),$$

$$\textit{predicted value: } \boldsymbol{y}_{i,t} = g_y(\boldsymbol{w}_y \boldsymbol{s}_{\boldsymbol{i,t}} + \boldsymbol{\beta}_y),$$

where $\circ$ is the element-wise product operator, $\boldsymbol{s_{i,0}} = \vec{\boldsymbol{0}}$ and $\boldsymbol{c_{i,0}} = \vec{\boldsymbol{0}}$. Assuming that there are α nodes in the hidden layer, the weight matrices $\boldsymbol{w}_f$, $\boldsymbol{w}_I$, $\boldsymbol{w}_o$, $\boldsymbol{w}_m$ have dimensions $\alpha \times \alpha$, and $\boldsymbol{u}_f$, $\boldsymbol{u}_I$, $\boldsymbol{u}_o$, $\boldsymbol{u}_m$ have dimensions $\alpha \times J$; the bias vectors $\boldsymbol{\beta}_f$, $\boldsymbol{\beta}_I$, $\boldsymbol{\beta}_o$, $\boldsymbol{\beta}_m$ have dimensions $\alpha \times 1$, and $\boldsymbol{w}_y$ and $\boldsymbol{\beta}_y$ for predictions have dimensions $\alpha \times 1$ and $K \times 1$, respectively. The functions $g_\cdot$ are the activation functions for the respective calculations where usually, g_f, g_I, g_o are sigmoid functions, and g_m, g_s are hyperbolic tangent functions (Gers et al., 2000; Gers et al., 2001) . In `R` (R Core Team, 2020), the package `keras` (Chollet et al., 2015) can be used to construct a recurrent layer with LSTM by using the pipeline `'layer_lstm()'`.

Masking and Padding

To prepare the longitudinal covariates for processing by a recurrent neural network, the observations for each subject need to be arranged in a $T \times p$ matrix where T is the maximum timestep of all subjects and p is the number of covariates. When the number of measurements varies by subject, a technique called "masking and padding" can be used (Gardner et al., 2018). The padding process allows placeholder values (usually 0's or 1's) to be filled into the timesteps without observations, and the masking process allows the recurrent layers to skip processing of these placeholder values. Padding can be done by basic `R` (R Core Team, 2020) operations, and masking can be done by the pipeline `'layer_masking'` in the package `keras` (Chollet et al., 2015).

2.3.3 Training the Neural Network

The model training process of the neural network aims to optimize the weights to best fit the training data. When building a neural network, different parameter settings need to be considered, for example, the type of each layer, the number of nodes in each layer, the activation function of each layer, the number of iterations, the type of optimizer, etc. Prior to the training of the actual neural network, hyperparameter tuning, the process that seeks to find the best combination of parameter settings for the neural network, must be performed. In this thesis, due to limited computer resources, the parameters that are tuned in the data analysis and simulation are limited to the number of nodes, dropout rates between the layers, and the activation functions in the layers. Table 2.1 refers to common activation functions that are available in `Python` (Python Core Team, 2020) library `Keras` (Chollet et al., 2015).

As this thesis is about binary classification, the loss function chosen is the binary cross-entropy loss (Murphy, 2012):

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right],$$

where y_i is the observed binary response variable, $\hat{y}_i$ is the predicted value of y_i, $i = 1, 2, ..., N$.

The stochastic optimizer used in this thesis is the adaptive moment estimation, or Adam (Kingma and Ba, 2014), which has been widely used since its implementation. The algorithm of Adam is given by:

$$\phi_\omega^{(n)} \leftarrow b_1\phi_\omega^{(n-1)} + (1-b_1)\nabla_\omega L^{(n-1)},$$

$$\psi_\omega^{(n)} \leftarrow b_2\psi_\omega^{(n-1)} + (1-b_2)(\nabla_\omega L^{(n-1)})^2,$$

$$\hat{\phi}_\omega^{(n)} \leftarrow \phi_\omega^{(n)}/(1-b_1^n),$$

$$\hat{\psi}_\omega^{(n)} \leftarrow \psi_\omega^{(n)}/(1-b_2^n),$$

Table 2.1: Common activation functions available in `Keras` (Chollet et al., 2015). ReLU refers to rectified linear unit. Tanh refers to hyperbolic tangent. SELU refers to scaled exponential linear unit. ELU refers to exponential linear unit. PReLU refers to parametric rectified linear unit. LeakyReLU refers to leaky rectified linear unit.

Activation	**Equation**	**Range**
ReLU	$g(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$	[0,∞)
Sigmoid	$g(x) = \frac{1}{1+\exp(-x)}$	(0,1)
Softmax	$g_i(x) = \frac{\exp(x_i)}{\sum_{j=1}^{J} \exp(x_j)}$	(0,1)
SoftPlus	$g(x) = \ln(1 + \exp(x))$	(0,∞)
Softsign	$g(x) = \frac{x}{1+\lvert x \rvert}$	(-1,1)
Tanh	$g(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	(-1,1)
SELU	$g(x) = \lambda \begin{cases} \alpha(\exp(x) - 1), & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}, \alpha = 1.6733, \lambda = 1.0507$	(-$\lambda\alpha, \infty$)
ELU	$g(x) = \begin{cases} \alpha(\exp(x) - 1), & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}, \alpha > 0$	(-α, ∞)
Exponential	$g(x) = \exp(x)$	(0, ∞)
PReLU	$g(x) = \begin{cases} \alpha x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}, \alpha > 0$	(-∞, ∞)
LeakyReLU	$g(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases},$	(-∞, ∞)

$$\omega^{(n)} \leftarrow \omega^{(n-1)} - \rho\hat{\phi}_{\omega}^{(n)}/(\sqrt{\hat{\psi}_{\omega}^{(n)}} + \xi),$$

where b_1 and b_2 are the exponential decay rates for the moment estimates, $\nabla_{\omega} L^{(n-1)}$ is the gradients of the loss function with respect to the weights ω at training iteration $n-1$, $\phi_{\omega}^{(n)}$ and $\psi_{\omega}^{(n)}$ are the updated first and second moment estimates at training iteration n, respectively, $\hat{\phi}_{\omega}^{(n)}$ and $\hat{\psi}_{\omega}^{(n)}$ are the unbiased first and second moment estimates at training iteration n, and $\omega^{(n)}$ is the updated weights at training iteration n, ρ is the learning rate, and ξ is a small scalar, $n = 1, 2, ..., E$. The default and initial values in `Python` (Python Core Team, 2020) library `Keras` (Chollet et al., 2015) are $b_1 = 0.9, b_2 = 0.999, \rho = 0.001, \xi = 10^{-7}, \phi_{\omega}^{(0)} =$

$\vec{0}, \psi_{\omega}^{(0)} = \vec{0}$, and $\omega^{(0)}$ is the random generated initial weights vector.

The evaluation metric used to select the best trained neural network is the loss of the validation set.

Chapter 3

Evaluation Methods

3.1 Training and Test Sets

To evaluate the performance of the models, each dataset will be split into two subsets - the subset for model training is called the training set, and the subset for testing the trained model is called the test set. In this thesis, the four types of models that are fit for the two datasets follow the same evaluation procedure. Firstly, the dataset is randomly split into two part by 70% : 30% ratio, where the two parts become training and test sets, respectively. Each of the models is then fit on the training set. For the neural networks, a validation set with 20% of the training set data is also randomly split from the training set at each iteration of the training process. The validation loss which measures the difference between the observed response in the 20% validation set and the predicted response in the validation set is used to select the best hyperparameter setting of a neural network.

In order to perform classification, an optimal probability threshold c^* that maximizes the lower of either the training set sensitivity (true positive rate) or specificity (true negative

rate) for each model is calculated as

$$c^* = \arg\max_c \min\left(P(p_i > c|y_i = 1), P(p_i \leq c|y_i = 0)\right) \forall i.$$

The predicted probability of event occurrence and binary outcome for test set, p_i^* and y_i^*, are then calculated as

$$p_i^* = \exp(x_i'\hat{\beta})/(1 + \exp(x_i'\hat{\beta})),$$

$$y_i^* = \begin{cases} 1, & \text{if } p_i^* \geq c^* \\ 0, & \text{otherwise.} \end{cases}$$

Lastly, p_i^* and y_i^* are used to calculate sensitivity, specificity, AUC, and BS for the test set, which also help to determine the prediction power of the models.

All models were fit using SHARCNET (Compute Canada) Graham cluster, which features $41,548$ CPU cores, 520 GPUs, approximately 20 Petabytes of storage, and 5 Petabytes of memory (RAM). For the computational tasks in this thesis, a unified set-up of 20 hours of task time limit, 32 CPU cores, 64GB (Gigabyte) of RAM was configured. This configuration is considered sufficient for the data sizes in this thesis, and by default, `tensorflow` (Allaire and Tang, 2020) uses as many cores as possible during the model training.

3.2 Akaike Information Criterion

The Akaike information criterion (Akaike, 1974), or AIC, was developed to measure the expected value of Kullback-Leibler (K-L) discrepancy (Kullback, 1959) between the real model, g, of the data X, and the fitted model, f. The AIC is defined as:

$$AIC = -2\log\left(\hat{L}(\boldsymbol{\theta})\right) + 2k, \tag{3.1}$$

where $\boldsymbol{\theta}$ is the parameter vector, k is the number of parameters, and $\hat{L}$ represents the maximum likelihood of a fitted statistical model for the data. Here $2k$ represents a penalty term for adding parameters. When parameters are added to the model, the maximum likelihood will increase, therefore the negative log-likelihood will decrease. To prevent the model from over-fitting, AIC balances the goodness of fit and the number of parameters. For a dataset with a large sample size, complex models will be preferred by AIC (Forster and Sober, 1994). The K-L discrepancy of f relative to g is defined as (Kullback, 1959):

$$d_{KL}(f|g) = E_g\left(\log\left(\frac{g(X)}{f(X)}\right)\right), \tag{3.2}$$

where E_g represents the expectation with respect to the real model g.

Stepwise AIC is a common model selection procedure that looks for the best model by decreasing the AIC throughout the process of adding or removing covariates. This paper will use a backward stepwise procedure that starts with the full model. At each step, the reduced model with one less covariate and smallest AIC is kept, until dropping of any covariate cannot further decrease the AIC. The `stepAIC` function in `R` (R Core Team, 2020) package `MASS` (Venables and Ripley, 2002) is used to perform stepwise AIC procedure.

3.3 The Confusion Matrix

The confusion matrix, also known as an error matrix, is a commonly used tool to evaluate the classification performance of a machine learning algorithm (Stehman, 1997). The confusion matrix is a special 2×2 contingency table where each row represents the instances in an actual class, and each column represents the instances in a predicted class (Powers, 2011). Table 3.1 shows a confusion matrix, where TP, FN, FP, and TN represent true positive, false negative, false positive, and true negative, respectively. True positive and true negative

mean the predicted positive or negative class matches the observed class for a certain subject.

Table 3.1: Typical Confusion Matrix

		Predicted Class: Positive	Predicted Class: Negative	Predicted Class: Total
Actual Class	Positive	TP	FN	$TP + FN$
	Negative	FP	TN	$FP + TN$
	Total	$TP + FP$	$FN + TN$	$TP + FN + FP + TN$

The overall accuracy of the model, AC (Powers, 2011), is the ratio of correct predictions to the total number of subjects, which is

$$AC = \frac{TP + TN}{TP + FN + FP + TN}. \tag{3.3}$$

Furthermore, the ratio of the number of correct positive predictions to the actual number of positive subjects, and the ratio of the number of correct negative predictions to the number of actual negative subjects, are known as true positive rate (TPR) and true negative rate (TNR) (Powers, 2011), or sensitivity and specificity, respectively, which are

$$TPR = \frac{TP}{TP + FN},$$

$$TNR = \frac{TN}{TN + FP}.$$

Depending on the costs of a mis-classified positive or negative subject, the importance of sensitivity and specificity can vary.

3.3.1 The Receiver Operating Characteristic Curve

The receiver operating characteristic (ROC) curve is a two dimensional plot that illustrates the classification power of a model as the discrimination threshold, c^*, varies. It is created by plotting sensitivity (TPR) against 1−specificity (FPR) under different thresholds (Fawcett,

2006).

The AUC measures the total area under the ROC curve. It is always between 0 and 1. The larger the AUC, the higher the overall accuracy of the model (Mandrekar, 2010).

3.4 The Brier Score

The squared error loss, also known as Brier score, or BS (Brier, 1950) is another commonly used model evaluation method for classification, which is defined as

$$BS = \frac{1}{N}\sum_{i=1}^{N}(p_i^* - y_i)^2,$$

where N is the number of subjects, p_i^* is the predicted probability of event occurrence for the i-th subject, $y_i = 1$ if the event occurs for subject i, and $y_i = 0$ otherwise. BS varies from 0 to 1, and lower BS means a lower error.

Chapter 4

Data Analysis

4.1 The Liver Dataset

The first dataset used for this thesis, the `liver` dataset, is from a liver cirrhosis treatment trial on 488 subjects with longitudinal covariates and a time-to-event response, survival time. It is available from `R` (R Core Team, 2020) package `joineR` (Philipson et al., 2018) and was initially taken from Andersen et al. (1993). The original purpose of the trial was to study the effect of a drug on treating liver cirrhosis by comparing the survival time of subjects who were treated with the drug versus those who were not. The covariates included in the dataset used for this thesis are: the value of the repeatedly measured prothrombin index (numeric), the time of each prothrombin measurement in years since the start of the study (numeric), the treatment indicator (binary). The response variable is the time of censoring or death in years (numeric) and the censoring indicator (binary).

Table 4.1 summarizes the subjects' survival times by 1-year, 5-year and 10-year categories. For this thesis, 1-year classification will be conducted. The response variable **died** is coded as 0 if the subject survived more than 1 year and 1 if the subject died within 1 year. The final data used in this thesis includes data for 441 subjects. The number of measurements

per subject has a mean of 3.09, a standard deviation of 0.84, a median of 3, a maximum of 6 and a minimum of 1. Tables 4.2 and 4.3 show the summary of the descriptive statistics for numeric and categorical variables in the final analyzed data, respectively.

Table 4.1: Summary of the subjects' 1-year, 5-year, 10-year mortality for the liver dataset

Time	**Survived**	**Died**	**Censored**
1 Year	332	109	47
5 Years	166	225	97
10 Years	17	289	182

Table 4.2: Means, medians, standard deviations, minimums and maximums of the numeric variables for the liver dataset, these variables are also time-varying variables which have 1364 observations

Variable	**Mean**	**Median**	**Sd.**	**Min.**	**Max.**
prothrombin	73.59	73.00	26.22	8	176
time	0.33	0.27	0.30	0	1

Table 4.3: Summary of the categorical variable for the liver dataset, this variable is also a time-fixed variable which has 441 observations

Variable		
treatment	No: 220	Yes: 221

4.1.1 Logistic Regression for the Liver Dataset

The built-in function `glm` (R Core Team, 2020) in `R` (R Core Team, 2020) will be used to fit the logistic regression model. The model is

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i},$$

where x_{1i} and x_{2i} represent the baseline covariates prothrombin and treatment, respectively, and p_i is the probability of survival for more than 1 year for the i-th subject, $i = 1, 2, ..., 441$. The response variable,

$$y_i = \begin{cases} 1, & \text{if the } i\text{-th subject survived for more than 1 year} \\ 0, & \text{otherwise,} \end{cases}$$

is assumed to follow a Bernoulli distribution with parameter p_i. Table 4.4 shows the summary of the fitted logistic regression model for the liver data. The optimal threshold c^* for this dataset is equal to 0.2484948. This value will be used later to make predictions and assess the performance of the model.

Table 4.4: Summary of the logistic regression model for the liver dataset

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	0.463	0.447	0.300
prothrombin	-0.026	0.006	0.000
treatment	0.124	0.276	0.653

4.1.2 Two-Stage Model for the Liver Dataset

`R` (R Core Team, 2020) functions `lme` (Pinheiro et al., 2020) and `glm` (R Core Team, 2020) can be used to fit the two stages of the joint model - the linear mixed effects submodel and the logistic regression submodel, respectively. Firstly, prediction of the random effect, $\hat{U}_i$, can be made by the linear mixed effects submodel. Secondly, the probability that the event occurs, $\hat{p}_i$, can be predicted by the logistic submodel. Lastly, the occurrence of the event, $\hat{y}_i$, can be predicted by comparing $\hat{p}_i$ and the optimum threshold.

For the liver data, let W_{ij} denote the prothrombin measurement on the i-th subject at time t_{ij}, $i = 1, 2, ..., 441$. Let x_i represent whether the i-th subject was given treatment (0

for placebo, 1 for treatment). Let p_i denote the probability that the i-th subject will die within 1 year. The two-stage model (Horrocks and van Den Heuvel, 2009) is

$$W_{ij} = \alpha_0 + \alpha_1 t_{ij} + \alpha_2 x_i + U_i + \epsilon_{ij}, \epsilon_{ij} \sim N(0, {\sigma_0}^2), U_i \sim N(0, {\sigma_1}^2)$$

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_i + \gamma_1 \hat{U}_i.$$

Table 4.5 and Table 4.6 show the results of fitting the linear mixed effects submodel and the logistic regression submodel, respectively. The optimal classification threshold for later performance assessment is $c^* = 0.2424432$.

Table 4.5: Summary of the linear mixed effects submodel for the liver dataset

Fixed-Effects	**Estimate**	**Std. Error**	**P-value**
intercept	67.304	1.807	0.000
time	7.929	1.961	0.000
treatment	7.921	2.452	0.001
Random-Effects	**Estimate**		
σ_1	19.004		

Table 4.6: Summary of the logistic regression submodel for the liver dataset

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	-1.461	0.216	0.000
treatment	0.165	0.288	0.565
$\hat{U}_i$	-0.056	0.010	0.000

4.1.3 Feed-forward Neural Network for the Liver Dataset

For the liver data, a deep neural network with input layer, three hidden layers and output layer is considered sufficient due to this data's small size and small number of covariates. Hyperparameter tuning on number of nodes, dropout rates, and activation functions were

performed for each layer to find the best model. Table 4.7 shows the network structure and values considered for each parameter. The whole training set was used for each iteration where 20% of the training set was randomly selected as a validation set, which varies iteration by iteration. There are 2 hyperparameter settings for the number of nodes, the dropout rates and activation functions for each hidden layer, and the activation function for the output layer. This leads to a total of 1024 (2^{10}) combinations of hyperparameter settings.

Table 4.7: Feed-forward neural network structure for the liver dataset with hyperparameter values considered

Layer	**# nodes**	**Dropout Rate**	**Activation**
Input Layer	2	-	-
First Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Second Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Third Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Output Layer	2	-	Sigmoid, Softmax

Table 4.8: Hyperparameter settings for the 10 feed-forward neural networks with lowest validation loss for the liver dataset. Validation loss refers to the cross-entropy loss for the validation set, and loss refers to the cross-entropy loss for the training set

# Nodes Hidden Layer 1, 2, 3	**Dropout Rates Hidden Layer 1, 2, 3**	**Activation Output Layer**	**Validation Loss**	**Loss**
32, 32, 32	0.3, 0.2, 0.3	Softmax	0.2520	0.6385
32, 16, 16	0.2, 0.2, 0.3	Softmax	0.2617	0.7129
32, 32, 32	0.3, 0.2, 0.2	Sigmoid	0.2671	0.6373
16, 32, 32	0.3, 0.2, 0.2	Sigmoid	0.2779	0.6306
32, 32, 16	0.2, 0.2, 0.2	Softmax	0.2829	0.6478
32, 16, 16	0.2, 0.3, 0.2	Sigmoid	0.2867	0.6170
16, 32, 32	0.3, 0.2, 0.2	Softmax	0.2873	0.6724
32, 32, 32	0.3, 0.3, 0.3	Sigmoid	0.2879	0.6332
16, 32, 32	0.2, 0.3, 0.2	Softmax	0.2929	0.6406
16, 32, 32	0.2, 0.3, 0.3	Softmax	0.2935	0.6411

The best network was selected from the networks with lowest validation set loss. Table

4.8 shows parameter settings and evaluation metrics of the 10 networks with lowest validation set loss. All of the ten models use ReLU, ReLU and Sigmoid activation functions for the three hidden layers, respectively. The neural network with lowest validation loss, which appears in the first row of the table, was selected to be the best model. This model was then used to make predictions to the test set as well as calculating the classification threshold according to the training set. The optimal classification threshold is $c^* = 0.2244313$.

4.1.4 Recurrent Neural Network for the Liver Dataset

A recurrent neural network with input layer, masking layer, recurrent layer with LSTM, and output layer is considered for this dataset. Hyperparameter values of number of nodes, dropout rates, and activation functions for the recurrent and output layers, were considered to find the best model. Table 4.9 shows the network structure and values considered for each parameter. For this neural network, there are 3 hyperparameter settings for the number of nodes, the dropout rates for the recurrent layer, and for the activation functions for the output layer. This leads to a total of 27 (3^3) combinations of hyperparameter settings.

Table 4.9: Recurrent neural network (with LSTM) structure for the liver dataset with hyperparameter values considered

Layer	**# nodes**	**Dropout Rate**	**Activation**
Input Layer	3	-	-
Masking Layer	-	-	-
Recurrent Layer with LSTM	8, 16, 32	0.2, 0.3, 0.4	tanh & Sigmoid
Output Layer	2	-	ReLU, Sigmoid, Softmax

The best network was selected from the networks with lowest validation set loss. Table 4.10 shows parameter settings and evaluation metrics of the 4 networks with lowest validation set loss, the best model appears in the first row of the table. This model was then used to make predictions to the test set as well as calculating the classification threshold according

to the training set. The optimal classification threshold is $c^* = 0.3537044$.

Table 4.10: Hyperparameter settings for the 4 recurrent neural networks (with LSTM) with lowest validation loss for the liver dataset. Validation loss refers to the cross-entropy loss for the validation set, and loss refers to the cross-entropy loss for the training set

# Nodes	Dropout Rates	Activations	Validation Loss	Loss
32	0.2	Softmax	0.3318	0.5383
32	0.3	Softmax	0.3410	0.5359
16	0.2	Sigmoid	0.3424	0.5936
32	0.4	Sigmoid	0.3430	0.5851

4.1.5 Model Evaluation for the Liver Dataset

To compare the prediction performance of the models, the sensitivity, specificity, AUC, and Brier Score are calculated, as Table 4.11 shows.

Table 4.11: Model performance evaluation for the liver dataset

Model	Sensitivity	Specificity	AUC	Brier Score
Baseline Logistic	0.806	0.646	0.748	0.176
Two-Stage	0.750	0.688	0.802	0.157
Feed-forward Neural Network	0.806	0.625	0.751	0.198
Recurrent Neural Network (LSTM)	0.861	0.333	0.626	0.205

Unsurprisingly, the two-stage model which used all the information in the dataset performed better than those models that only used information at the first measurement time, namely baseline logistic regression model and the feed-forward neural network model. It has the highest specificity, AUC, and the lowest Brier score although not the highest sensitivity. However, given that the recurrent neural network also used all the information, it was expected to outperform the baseline logistic and feed-forward neural network, which is not the case. One possible reason could be over-fitting of the training set. It is also noted that the

performance of the baseline logistic model and feed-forward neural network are comparable. Overall, for the `liver` dataset, the two-stage joint model was preferred and no obvious advantage of neural networks was found.

4.2 The PBC2 Dataset

The second dataset, `pbc2`, used for this thesis is a primary biliary cirrhosis data on 312 subjects with longitudinal covariates and a time-to-event response, survival time, recorded in a randomized trial conducted by Mayo Clinic (Fleming and Harrington, 1991). It is available in `R` (R Core Team, 2020) package `JMbayes` (Rizopoulos, 2016). The original purpose of the data was to study the effect of the drug D-penicillamine (DPCA) on treating primary biliary cirrhosis by comparing the survival time of subjects who were treated with those who were not. The data includes 3 time-fixed and 13 time-varying covariates. Time-fixed covariates were age in years at the beginning of the trial (numeric), sex (binary) and the treatment indicator (binary). Time-varying covariates were recorded at every visit, and includes measurements of serum bilirubin in milligram per deciliter (numeric), serum cholesterol in milligram per deciliter (numeric), albumin in milligram per deciliter (numeric), alkaline phosphatase in units per liter (integer), serum glutamic-oxaloacetic transaminase (SGOT) in units per milliliter (numeric), platelets per cubic milliliter per thousand (integer), prothrombin time in seconds (numeric), histologic stage of disease (4-level); and whether the patient had ascites (binary), hepatomegaly (binary), spiders (binary), edema (3-levels), the time of each visit in years (numeric). The response variable is the time between registration and earlier of death, transplantation, or censoring in years (numeric) and the status indicator (3-level).

Table 4.12 summarizes the subjects' survival times by 1-year, 5-year and 10-year categories. For this thesis, a 5-year classification will be conducted. The response variable died

is coded as 0 if the subject survived more than 5 years and 1 if the subject died within 5 years. The 22 subjects who got transplanted before the end of 5-year were omitted as their classification results remain unknown. Those who got transplanted after the end of 5-year were coded survived. There is a significant number of missing values in this dataset, mostly in the covariate serChol, which was therefore dropped from the analysis. Furthermore, 82 more observations were dropped due to missing values elsewhere. The final data used in this thesis includes 290 subjects and it is split into training and test sets as in section 3.1. The number of measurements per subject has a mean of 4.50, a standard deviation of 1.94, a median of 5, a maximum of 7 and a minimum of 1. Tables 4.13 and 4.14 show the summary of the descriptive statistics for numeric and categorical variables, respectively.

Table 4.12: Summary of the subjects' 1-year, 5-year, 10-year mortality for the PBC2 dataset

Time	**Survived**	**Died**	**Censored**
1 Year	290	22	0
5 Years	202	88	22
10 Years	51	131	130

Table 4.13: Means, medians, standard deviations, minimums and maximums of the numeric variables for the PBC2 dataset, age is a time-fixed variable which has 290 observations, and the others are time-varying variables which have 1304 observations

Variable	**Mean**	**Median**	**Sd.**	**Min.**	**Max.**
age	50.46	50.30	10.51	26.28	78.44
year	1.59	1.03	1.49	0	4.99
serBilir	3.26	1.3	4.74	0.1	36
albumin	3.8	3.52	0.48	1.17	8.01
alkaline	1504.0	1157.0	1337.60	130	13862
SGOT	124.7	111.6	71.92	6.2	918
platelets	244.42	240.0	99.33	41	991
prothrombin	10.81	10.60	1.32	9	31.80

Table 4.14: Summary of the categorical variables for the PBC2 dataset, drug and sex are time-fixed variables which have 290 observations, and the others are time-varying variables which have 1304 observations

Variable				
drug	placebo: 143	D-penicil: 147		
sex	male: 34	female: 256		
ascites	No: 1199	Yes: 105		
hepatomegaly	No: 676	Yes: 628		
spiders	No: 904	Yes: 400		
edema	No: 993	Yes: 311		
histologic	1: 78	2: 211	3: 446	4: 569

4.2.1 Logistic Regression for the PBC2 dataset

The built-in function `glm` (R Core Team, 2020) in `R` (R Core Team, 2020) was be used to fit the logistic regression model for the baseline observations. The full model is considered:

$$\log\left(\frac{p_i}{1-p_i}\right) = \mathbf{x}_i^{'}\boldsymbol{\beta},$$

where $\mathbf{x}_i$ represents the baseline covariates vector including intercept for the i-th subject, and $\boldsymbol{\beta}$ represents the regression coefficients vector. Here p_i is the probability of survival for more than 5 years for the i-th subject, $i = 1, 2, ..., 290$. The response variable,

$$y_i = \begin{cases} 1, & \text{if the } i\text{-th subject survived for more than 5 years} \\ 0, & \text{otherwise,} \end{cases}$$

is assumed to follow a Bernoulli distribution with parameter p_i. Table 4.15 shows the summary of the baseline logistic full model for the PBC2 data. The optimum threshold for classification is $c^* = 0.3548031$. Table 4.16 shows the summary of the reduced baseline logistic model selected by backward stepwise AIC using `R` (R Core Team, 2020) package `MASS` (Venables and Ripley, 2002) for the PBC2 data. The optimum threshold for classification is

$c^* = 0.3214004$.

Table 4.15: Summary of the baseline logistic regression model for the PBC2 dataset

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	-9.964	4.540	0.028
drug	0.035	0.466	0.940
age	0.062	0.026	0.015
sex(female)	-0.430	0.677	0.526
ascites	0.841	1.040	0.419
hepatomegaly	0.337	0.553	0.541
spiders	0.320	0.517	0.536
edema	0.636	0.577	0.270
serBilir	0.479	0.133	0.000
albumin	-0.218	0.619	0.725
alkaline	0.000	0.000	0.675
SGOT	0.004	0.004	0.331
platelets	-0.003	0.003	0.327
prothrombin	0.289	0.294	0.326
histologic	0.665	0.385	0.084

Table 4.16: Summary of the baseline logistic regression model selected by backward stepwise AIC for the PBC2 dataset

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	-7.927	1.878	0.000
age	0.065	0.023	0.004
edema	0.796	0.519	0.125
serBilir	0.595	0.119	0.000
platelets	-0.004	0.003	0.134
histologic	0.903	0.316	0.004

4.2.2 Two-Stage Model for the PBC2 dataset

For the `pbc2` data, let W_{ij} denote the prothrombin measurement on the i-th subject at time t_{ij}, $i = 1, 2, ..., 290, j = 1, 2, ..., m_i$, where m_i is the number of measurement taken for subject i. Let x_i represent whether the i-th subject was given a drug (0 for placebo, 1 for D-penicil).

Let p_i denote the probability that the i-th subject dies within 5 years. The following two-stage model (Horrocks and van Den Heuvel, 2009) consists of a linear mixed-effect model for longitudinal prothrombin measurements and a logistic model:

$$W_{ij} = \boldsymbol{\delta} X_i + \boldsymbol{\zeta} Z_{ij} + U_i + \epsilon_{ij}, \epsilon_{ij} \sim N(0, {\sigma_0}^2), U_i \sim N(0, {\sigma_1}^2)$$

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 v_{i1} + \beta_2 v_{i2} + \beta_3 v_{i3} + \gamma_1 \hat{U}_i,$$

where v_{i1}, v_{i2}, v_{i3} are covariates **drug**, **sex**, **age** for the i-th subject, X_i is the vector of time-fixed covariates for the i-th subject, Z_{ij} is the vector of time-varying covariates at the j-th observation for the i-th subject, U_i and $\hat{U}_i$ are the random effect and its estimates, respectively. Table 4.17 and Table 4.18 show the result of the longitudinal submodel and the logistic regression submodel, respectively. The optimal threshold for the prediction is $c^* = 0.3197601$.

Variable selection for both submodels was performed by backward stepwise AIC using `R` (R Core Team, 2020) package `MASS` (Venables and Ripley, 2002). Table 4.19 and Table 4.20 show the summary of both reduced submodels selected. The optimum threshold for classification is $c^* = 0.3250344$.

4.2.3 Feed-forward Neural Network for the PBC2 dataset

For the `pbc2` dataset, a deep neural network similar to the one for the `liver` dataset was considered. It features an input layer, three hidden layers and an output layer. Hyperparameter tuning on number of nodes, dropout rates, and activation functions were performed for each layer to find the best model. Table 4.21 shows the network structure and values considered for each parameter. The whole training set was used for each training iteration where 20% of the training set was randomly selected as a validation set, which varies iteration by

Table 4.17: Summary of the linear mixed effects submodel for the PBC2 dataset

Fixed-Effects	**Estimate**	**Std. Error**	**P-value**
Intercept	12.074	0.542	0.000
drug	-0.012	0.100	0.903
age	0.002	0.005	0.667
sex(female)	-0.293	0.153	0.058
year	-0.011	0.025	0.670
ascites	0.282	0.156	0.071
hepatomegaly	-0.161	0.089	0.072
spiders	0.420	0.096	0.000
edema	0.154	0.106	0.145
serBilir	0.054	0.012	0.000
albumin	-0.356	0.090	0.000
alkaline	-0.000	0.000	0.086
SGOT	0.000	0.001	0.753
platelets	-0.001	0.000	0.029
histologic	0.050	0.055	0.368
Random-Effects	**Estimate**		
σ_1	0.484		

Table 4.18: Summary of the logistic regression submodel for 2-stage model

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	-3.657	1.033	0.000
drug	-0.006	0.316	0.985
age	0.061	0.017	0.000
sex	-0.220	0.483	0.649
$\hat{U}_i$	0.620	0.451	0.169

iteration. There are 2 hyperparameter settings for the number of nodes, the dropout rates and activation functions for each hidden layer, and the activation function for the output layer as well. This leads to a total of 1024 (2^{10}) combinations of hyperparameter settings.

The best network was selected from the networks with lowest validation set loss. Table 4.22 shows parameter settings and evaluation metrics of the 10 networks with lowest validation set loss. All of the 10 networks use Sigmoid as the activation function for the first

Table 4.19: Summary of the stepwise AIC selected linear mixed effect submodel for the PBC2 dataset

Fixed-Effects	**Estimate**	**Std. Error**	**P-value**
Intercept	12.387	0.367	0.000
sex(female)	-0.292	0.150	0.053
ascites	0.290	0.155	0.062
hepatomegaly	-0.138	0.087	0.112
spiders	0.422	0.095	0.000
edema	0.158	0.104	0.130
serBilir	0.055	0.011	0.000
albumin	-0.369	0.087	0.000
alkaline	-0.000	0.000	0.094
platelets	-0.001	0.000	0.021
Random-Effects	**Estimate**		
σ_1	0.491		

Table 4.20: Summary of the stepwise AIC selected logistic regression submodel for the PBC2 dataset

Coefficients	**Estimate**	**Std. Error**	**P-value**
Intercept	-3.884	0.863	0.000
age	0.061	0.016	0.000
$\hat{U}_i$	0.650	0.443	0.143

Table 4.21: Feed-forward neural network structure for the PBC2 dataset with hyperparameter tuning grids

Layer	**# nodes**	**Dropout Rate**	**Activation**
Input Layer	15	-	-
First Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Second Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Third Hidden Layer	16, 32	0.2, 0.3	ReLU, Sigmoid
Output Layer	2	-	Sigmoid, Softmax

hidden-layer. The neural network with lowest validation loss, which appears in the first row of the table, was selected to be the best model. The optimal threshold for classification is $c^* = 0.4394991$.

Table 4.22: Hyperparameter settings for the 10 feed-forward neural networks with low validation loss for the PBC2 dataset. Validation loss refers to the cross-entropy loss for the validation set, and loss refers to the cross-entropy loss for the training set

# Nodes Hidden Layer 1, 2, 3	Dropout Rates Hidden Layer 1, 2, 3	Activation Hidden/Output Layer 2,3, Output	Validation Loss	Loss
16, 16, 32	0.2, 0.2, 0.3	ReLU, ReLU, Softmax	0.4330	0.5367
16, 32, 32	0.3, 0.2, 0.2	ReLU, ReLU, Sigmoid	0.4341	0.5453
16, 32, 32	0.3, 0.2, 0.3	ReLU, Sigmoid, Softmax	0.4464	0.6012
32, 32, 16	0.2, 0.2, 0.2	ReLU, Sigmoid, Softmax	0.4481	0.5884
16, 32, 16	0.3, 0.3, 0.2	ReLU, ReLU, Sigmoid	0.4490	0.5797
32, 32, 16	0.3, 0.3, 0.3	ReLU, ReLU, Sigmoid	0.4552	0.5658
32, 16, 32	0.2, 0.3, 0.3	ReLU, ReLU, Softmax	0.4564	0.5460
32, 32, 32	0.2, 0.2, 0.3	Sigmoid, ReLU, Softmax	0.4578	0.5495
32, 32, 16	0.3, 0.2, 0.3	ReLU, Sigmoid, Softmax	0.4586	0.5857
32, 32, 16	0.2, 0.3, 0.2	ReLU, ReLU, Sigmoid	0.4593	0.5132

4.2.4 Recurrent Neural Network for the PBC2 dataset

Similar to the recurrent neural network for the `liver` dataset, a recurrent neural network with input layer, masking layer, recurrent layer with LSTM, and output layer is considered for this dataset. Hyperparameter values of the number of nodes, dropout rates, and activation functions for the recurrent and output layers were considered to find the best model. Table 4.23 shows the network structure and values considered for each parameter. For this neural network, there are 3 hyperparameter settings for the number of nodes, the dropout rates for the recurrent layer, and for the activation functions for the output layer, which leads to a total of 27 (3^3) combination of hyperparameter settings.

The best network was selected from the networks with lowest validation set loss. Table 4.24 shows parameter settings and evaluation metrics of the 4 networks with lowest validation set loss. The best model appears in the first row of the table, this model was then used to make predictions to the test set as well as calculating the classification threshold according to the training set. The optimal classification threshold is $c^* = 0.3717531$.

Table 4.23: Recurrent neural network (with LSTM) structure for the PBC2 dataset with hyperparameter tuning grids

Layer	# nodes	Dropout Rate	Activation
Input Layer	15	-	-
Masking Layer	-	-	-
Recurrent Layer with LSTM	8, 16, 32	0.2, 0.3, 0.4	tanh & Sigmoid
Output Layer	2	-	ReLU, Sigmoid, Softmax

Table 4.24: Hyperparameter settings for the 4 recurrent neural networks (with LSTM) with low validation loss for the PBC2 dataset. Validation loss refers to the cross-entropy loss for the validation set, and loss refers to the cross-entropy loss for the training set

# Nodes	Dropout Rates	Activations	Validation Loss	Loss
8	0.4	Softmax	0.5294	0.6170
16	0.2	Sigmoid	0.5324	0.6422
32	0.4	Softmax	0.5341	0.6423
16	0.3	Sigmoid	0.5513	0.6608

4.2.5 Model Evaluation for the PBC2 dataset

To compare the prediction performance of the models, the sensitivity, specificity, AUC, and BS are calculated respectively, as Table 4.25 shows.

Table 4.25: Model performance evaluation for the PBC2 dataset

Model	Sensitivity	Specificity	AUC	Brier Score
Baseline Logistic	0.864	0.846	0.932	0.087
Baseline Logistic (AIC)	0.864	0.815	0.920	0.094
Two-Stage	0.636	0.585	0.655	0.183
Two-Stage (AIC)	0.591	0.600	0.638	0.187
Feed-forward Neural Network	0.636	0.662	0.714	0.184
Recurrent Neural Network (LSTM)	0.409	0.554	0.553	0.211

For the `pbc2` dataset, surprisingly, the baseline logistic model is the best model overall. It has the highest prediction sensitivity, specificity, AUC, and the lowest BS. Compared to the baseline logistic model, the reduced baseline logistic model selected by backward

stepwise AIC has slightly lower prediction performance. These two models out-performed both two-stage models. One possible reason is that both baseline logistic models have more covariates than the second stage of the two-stage models. The baseline logistic and reduced baseline logistic models include 15 and 5 covariates, respectively, while the second stage of the two-stage and reduced two-stage model only include 4 and 2 covariates, respectively. Furthermore, a highly predictive time-varying covariate, **serBilir**, was not included in the logistic submodel of the two-stage model, which may also lead to inaccurate predictions. For the neural networks, the feed-forward neural network has better prediction performance than the recurrent neural network with LSTM despite using only baseline information. Between the statistical models and the neural networks, the performance of the feed-forward neural network is lower than the baseline logistic models and higher than the two-stage models. Overall, there is no obvious advantage of the neural networks compared with the statistical models.

Chapter 5

Simulation Study

In this chapter, a simulation study is described that was conducted using simulated data with longitudinal covariates and binary response, by fitting and comparing the performances among baseline logistic regression model, 2-stage model, and feed-forward and recurrent neural network models in terms of their prediction BS (Brier, 1950), sensitivity, specificity, and AUC.

5.1 Generating Data

Inspired by the `liver` dataset, the simulated data consists of subject identifications $i, i = 1, 2, ..., N$, observation times $t_j, j = 1, 2, ..., T$, a time-varying response W_{ij}, a time-fixed covariate $x_i, x_i \sim Bernoulli(p_x)$, and the binary response variable $y_i, \sim Bernoulli(p_i)$. As all the subjects have the same observation times, the simulated data will be balanced. The longitudinal response W_{ij} is generated from a linear mixed model and binary response variable y_i is generated from the logistic model (Horrocks and van Den Heuvel, 2009):

$$W_{ij} = \alpha_0 + \alpha_1 t_j + \alpha_2 x_i + U_i + \epsilon_{ij}, \epsilon_{ij} \sim N(0, {\sigma_\epsilon}^2), U_i \sim N(0, {\sigma_U}^2)$$

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_i + \gamma_1 U_i,$$

$$\text{and} \begin{cases} P(y_i = 0|p_i) = 1 - p_i, \\ P(y_i = 1|p_i) = p_i, \end{cases}$$

where α_0, α_1 and α_2 are the fixed-effect parameters, U_i is the random effect, and ϵ_{ij} is the error term.

The whole data simulation procedure works as follow. The error ϵ_{ij} is generated from $N(0, {\sigma_\epsilon}^2)$, the random effect U_i is generated from $N(0, {\sigma_U}^2)$, and the time-fixed covariate x_i is generated from $Bernoulli(p_x)$ for $i = 1, 2, ..., N$ and $j = 1, 2, ..., T$. The time-varying covariate W_{ij} is then calculated based on t_j, x_i, U_i, and ϵ_{ij}, and the initial parameters $\alpha_0, \alpha_1, \alpha_2$ are based on the longitudinal submodel fitted to `liver` dataset. Then, a new longitudinal submodel will be fit to the generated data, to calculate the fitted random effect $\hat{U}_i$ so that the probability of the event of interest, p_i, can be calculated based on $x_i, \hat{U}_i$, and the initial parameters $\beta_0, \beta_1, \gamma_1$ that are based on the logistic submodel. Lastly, the binary response variable y_i is generated from $Bernoulli(p_i)$. Table 5.1 shows the values of the initial parameters for data simulations. Table 5.2 shows the variables generated during the data simulations. In this simulation study, the observation times t_j are evenly distributed from 0 to 1, $t_j = [1/6, 2/6, 3/6, 4/6, 5/6]$.

Table 5.1: Summary of the values for parameters in simulation study

Parameter	**Initial Value**
N	500
T	5
p_x	0.499
σ_ϵ	17.136
σ_U	18.983
α_0	66.247
α_1	10.343
α_2	7.290
β_0	-1.449
β_1	0.202
γ_1	-0.061

Table 5.2: Variables, random effects, and error terms generated in simulation study

Variable	**Distribution**
U_i	$N(0, {\sigma_U}^2)$
ϵ_{ij}	$N(0, {\sigma_\epsilon}^2)$
x_i	$Bernoulli(p_x)$
y_i	$Bernoulli(p_i)$

5.2 Simulation Design

This simulation study is a nested loop coded in `R` (R Core Team, 2020) which features 1000 iterations. At each iteration, a simulated dataset is generated, and it is split into training and test sets. A baseline logistic regression model, a two-stage model, a feed-forward neural network, and a recurrent neural network with LSTM, are fitted to the simulated training set. Due to limited computational resources, the hyperparameter settings found in the `liver` data analysis were kept for both neural networks, respectively (Table 4.8 and Table 4.10), and no hyperparameter tuning was performed in the simulation study. These models are

then used to make predictions to the simulated test set and BS (Brier, 1950), sensitivity, specificity, and AUC are recorded.

5.3 A Comparison of Models Using Simulated Data

Table 5.3 shows the mean and standard deviation of the predicted BS (Brier, 1950), sensitivity, specificity, and AUC of the four types of models.

Table 5.3: Model performance evaluation for simulated dataset

Model	**Mean Sens. (Sd.)**	**Mean Spec. (Sd.)**	**Mean AUC (Sd.)**	**Mean BS (Sd.)**
Baseline Logistic	0.639	0.643	0.697	0.172
	(0.0831)	(0.056)	(0.050)	(0.018)
Two-Stage	0.678	0.683	0.747	0.162
	(0.085)	(0.054)	(0.047)	(0.018)
Feed-forward Neural Network	0.594	0.593	0.665	0.188
	(0.117)	(0.099)	(0.069)	(0.019)
Recurrent Neural Network	0.457	0.546	0.539	0.199
	(0.102)	(0.081)	(0.038)	(0.020)

The results of the simulation study are similar to the `liver` data analysis. The two-stage model performed the best overall for all evaluation metrics. It has the highest average prediction sensitivity, specificity, AUC, and the lowest average Brier score. It also has the lowest standard deviation on specificity, and second lowest standard deviation on the rest of the evaluation metrics. The neural networks' performances remain relatively poor, especially for the recurrent neural network with the average prediction sensitivity below 0.5, and average AUC barely above 0.5.

Chapter 6

Ethical Implications

While artificial intelligence (AI), such as the deep neural network, provides a comprehensive solution in classification of data with binary response and longitudinal covariates, it is very important to take possible ethical implications into account. In this chapter, the following concerns that are related to the data analysis in the previous chapters will be discussed: safety, privacy, liability, opacity, bias, and climate change.

This paper focuses on the comparison of the performance of survivability classification between the statistical models and AI. Inappropriate and inaccurate models could result in misdiagnosis of a patient and put him/her at risk. On one hand, a necessary treatment might not be given to the patient if his/her predicted survivability is over-estimated. On the other hand, unnecessary treatment could be given to the patient if the predicted survivability is under-estimated, which also potentially affects their health. To prevent the above situations from happening, the models discussed should only be used as extra support for clinical decisions until they are demonstrated to be reliable.

Patients' privacy is also a significant concern as their personal health data are used for model training and validation. The datasets involved in this paper, `pbc2` and `liver`, are open sourced and free to use. Both of them exclude covariates that identify the patients by

randomly assigning identification numbers to them. Therefore, patients privacy was not a concern in this paper.

As previously mentioned, the models discussed in this paper are for the purpose of assisting medical diagnosis, which means they cannot be used by the patients to make their own medical decisions. Therefore, there will be no direct liability caused by these models. However, medical professionals who diagnose a patient assisted by these models could make wrong decisions as a result of inappropriate models, and could likely be held liable.

Compared to the statistical models, the opacity of the AI is a significant concern. The AI methods used in this thesis are feed-forward neural network and recurrent neural network, which have low interpretability. The users are not able to know the effects of covariates on the outcomes, and therefore could not analyze the significance of these covariates or the appropriateness of the models. Unlike the statistical models, the AI models cannot be statistically tested, and the only feedbacks are the performance metrics and tuned parameter settings. In another word, the AI models are "black-boxes" for the users.

The bias of the models is another ethical concerns. If models are developed on datasets that under-represent some groups, they can result in inaccurate predictions.For example, if certain groups of categorical covariates barely appears, the model will be inaccurate for those groups. The way to overcome this issue is to keep training the model with larger and more balanced datasets.

For the relatively small sample sizes (e.g., hundreds or few thousands) which were considered in data analysis and simulation sections in this thesis, the neural networks brought no obvious prediction advantages compared to traditional statistical methods. However, the hyperparameter tuning process was very time and power consuming. Depending on the tuning grids of the hyperparameters, the training of a three-hidden-layer feed-forward neural network could take several days of computing on a SHARCNET (Compute Canada) node with 32 CPU cores. Furthermore, adding more hidden layers could result in exponentially

increased training time and computing power, which brings significant concerns about the power usage, and contribute to climate change.

Overall, while safety, privacy and liability concerns remain low for the models discussed in this thesis, there are ethical concerns in terms of opacity, bias and climate change. While there is no obvious solution to increase the opacity of the neural networks, it is believed that the bias could be decreased by training the model using bigger and more balanced data.

Chapter 7

Conclusion and Further Work

In this thesis, the classification performance of two widely used neural networks, the feed-forward neural network and the recurrent neural network with long short-term memory, and two statistical approaches, logistic regression and the two-stage joint model (Horrocks and van Den Heuvel, 2009), were compared in terms of prediction sensitivity, specificity, AUC, and BS (Brier, 1950), on two clinical trial datasets with longitudinal covariates and a binary categorical response variable. Furthermore, a simulation study on data generated based on one of the real datasets, the `liver` dataset, was conducted to compare the performance of these models by the same evaluation metrics. While the recurrent neural networks and the two-stage models were fit using all the information from the datasets, the feed-forward neural networks and the baseline logistic regression were fit using only the information from the first measurement of each subject.

From the result of the data analysis and simulation, for such small datasets with less than 500 subjects, neither of the neural networks outperformed the best statistical models in any of the analyses. While the feed-forward neural networks have comparable prediction performance with the statistical models, the recurrent neural networks with LSTM performed worse than the others. Specifically, in the `liver` data analysis and the simulation study, the

two-stage joint models showed obvious advantages, and in the `pbc2` data analysis, the baseline logistic regression outperformed the two-stage model, probably due to the extra covariates included in the logistic model. Overall, the conventional statistical methods performed better than the neural networks for the problems that were discussed in this thesis.

In addition to the findings related to model performance, this thesis also brings up a concern related to the computational power usage for training the neural networks. While the data analysis by statistical models took seconds, it took hours and days for the neural networks. Overall, combining this concern with the disappointing performance of the neural networks that was discussed earlier, it is not suggested to use deep learning to perform classification in relatively small datasets.

Further Work

The response variables in this thesis were not time-varying variables. The methods used in this thesis could be compared for data with a time-varying response variable and time-varying covariates, referred to as "longitudinal data" in the statistical literature.

Due to limited computational resources, the hyperparameter tuning for the neural networks were limited to the activation functions, number of nodes, and dropout rates. To construct the best neural network, more hyperparameter types could be considered, for example, the stochastic optimizer, the number of iterations, the early-stop criteria, the size of mini-batch etc. Once again, tuning all these parameters could exponentially increase the model training time (i.e. weeks or months).

Further suggested work is to increase the simulation data size, to the scale of 10^5 subjects, to see whether the performance of the neural networks would improve under such a circumstance, as neural networks normally require large datasets for good performance. This thesis was not able to do so due to limited computational resources.

Another suggested further work is to fit the logistic regression and feed-forward neural

network with the same complete information that were used for the two-stage model and recurrent neural network with LSTM, instead of using only the baseline measurement. Note that in order for the feed-forward neural network to exploit the correlation between measurements for the same subject, subject ID could be included as a covariate. Doing so can provide a way to make more fair comparison among all four models.

Lastly, for fitting a two-stage joint model, methods of predicting U_i for subjects in the test set have not been investigated. An estimate was proposed in Section 2.2, but its properties have not been studied. This is a topic for further research.

Bibliography

[1] Akaike, H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control 19*, 6 (1974), 716–723.

[2] Allaire, J., and Tang, Y. *tensorflow: R Interface to 'TensorFlow'*, 2020. R package version 2.2.0.

[3] Andersen, R. J., Luu, H. A., Chen, D. Z., Holmes, C. F., Kent, M. L., Le Blanc, M., Williams, D. E., et al. Chemical and biological evidence links microcystins to salmon 'netpen liver disease'. *Toxicon 31*, 10 (1993), 1315–1323.

[4] Biganzoli, E., Boracchi, P., Mariani, L., and Marubini, E. Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach. *Statistics in Medicine 17*, 10 (1998), 1169–1186.

[5] Blundell, R., and Mátyás, L. Panel data analysis: An introductory overview. *Structural Change and Economic Dynamics 3*, 2 (1992), 291–299.

[6] Breslow, N. E., and Clayton, D. G. Approximate inference in generalized linear mixed models. *Journal of the American Statistical Association 88*, 421 (1993), 9–25.

[7] Brier, G. W. Verification of forecasts expressed in terms of probability. *Monthly Weather Review 78*, 1 (1950), 1–3.

[8] Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y., and Bengio, Y. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983* (2017).

[9] Cheng, M.-S., Sheu, J.-P., Van Cuong, N., and Kuo, Y. C. Mobility prediction at points of interest using many-to-one recurrent neural network. In *GLOBECOM 2020-2020 IEEE Global Communications Conference* (2020), IEEE, pp. 1–6.

[10] Choi, E., Schuetz, A., Stewart, W. F., and Sun, J. Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association 24*, 2 (2017), 361–370.

[11] Chollet, F., et al. Keras. https://github.com/fchollet/keras, 2015.

[12] Falissard, L., Fagherazzi, G., Howard, N., and Falissard, B. Deep clustering of longitudinal data. *arXiv preprint arXiv:1802.03212* (2018).

[13] Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.

[14] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., and Zettlemoyer, L. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640* (2018).

[15] Gers, F. A., and Schmidhuber, E. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks 12*, 6 (2001), 1333–1340.

[16] Gers, F. A., Schmidhuber, J. A., and Cummins, F. A. Learning to forget: Continual prediction with LSTM. *Neural Computation 12*, 10 (2000), 2451–2471.

[17] Horrocks, J., van Den Heuvel, M. J., et al. Prediction of pregnancy: a joint model for longitudinal and binary data. *Bayesian Analysis 4*, 3 (2009), 523–538.

[18] Hosmer, D. W., Lemeshow, S., and Sturdivant, R. X. *Applied logistic regression*, vol. 398. John Wiley & Sons, 2013.

[19] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Mandrekar, J. N. Receiver operating characteristic curve in diagnostic test assessment. *Journal of Thoracic Oncology 5*, 9 (2010), 1315–1316.

[21] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2020. R package version 3.1-144.

[23] Powers, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061* (2011).

[24] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

[25] Reddy, B. K., and Delen, D. Predicting hospital readmission for lupus patients: An rnn-lstm-based deep-learning methodology. *Computers in Biology and Medicine 101* (2018), 199–209.

[26] Stehman, S. V. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment 62*, 1 (1997), 77–89.

[27] Tolles, J., and Meurer, W. J. Logistic regression: relating patient characteristics to outcomes. *JAMA 316*, 5 (2016), 533–534.

[28] Venables, W. N., and Ripley, B. D. *Modern applied statistics with S*. Springer, 2002.

[29] Wang, C., Wang, N., and Wang, S. Regression analysis when covariates are regression parameters of a random effects model for observed longitudinal measurements. *Biometrics 56*, 2 (2000), 487–495.

[30] Zeger, S. L., Liang, K.-Y., and Albert, P. S. Models for longitudinal data: a generalized estimating equation approach. *Biometrics* (1988), 1049–1060.

[31] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. Dive into deep learning. *Unpublished Draft. Retrieved 19* (2019), 2019.

[32] Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. Self-attention generative adversarial networks. In *International Conference on Machine Learning* (2019), pp. 7354–7363.

Appendix A

Source Code

```
#####################################
#########Liver Data Analysis#########
#####################################

#######SHARCNET Job Submission#######

#!/bin/bash
#SBATCH --time=20:00:00
#SBATCH --account=def-jhorrock
#SBATCH --job-name=liver
#SBATCH --ntasks=32
#SBATCH --mem-per-cpu=2GB
#SBATCH --output=slurm-%A_%a.out
#SBATCH --mail-user=zchen23@uoguelph.ca
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
#SBATCH --mail-type=REQUEUE
#SBATCH --mail-type=ALL

module load nixpkgs/16.09 gcc/8.3.0 r/4.0.0 python/3.6
source tensorflow/bin/activate
Rscript ~/scratch/Data/liver/liver.R

#####################################

####~/scratch/Data/liver/liver.R####

##Working directory, packages and dataset
setwd("~/scratch/Data/liver")
```

```
library(keras)
use_session_with_seed(123)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(joineR)
library(pROC)
library(JM)
library(tensorflow)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(tfruns)
data(liver)
set.seed(123)
liver=data.frame(liver,NA,NA)
colnames(liver)=c("id","prothrombin","time","treatment",
                  "survival","cens","died","ric")

##Died=0 if survived 1 year, =1 if died within 1 year,
##=2 if censored
for (i in 1:2969) {
  if (liver[i,6] == 1&liver[i,5]<=1) liver[i,7]<-1
  else if (liver[i,5] >1) liver[i,7]<-0
  else liver[i,7]<-2
}
##Eliminate the censored data and the
##measurement time later than 1 year
liver<-liver[which(liver$died<2&liver$time<=1),]

##Generate training IDs
training.index <- sample(unique(liver$id),309, replace = FALSE)

##Training and test sets
train<-liver[which(liver$id %in% training.index),]
test<-liver[-which(liver$id %in% training.index),]

##Logistic regression with first observation
bltrain<-data.frame(NA,NA,NA)
colnames(bltrain)<-c("prothrombin","treatment","died")
bltest<-data.frame(NA,NA,NA)
colnames(bltest)<-c("prothrombin","treatment","died")
for (i in 1:955) {
  bltrain[i,]<-train[which(train$id==i),c(2,4,7)][1,]
}
bltrain<-bltrain[complete.cases(bltrain),]
for (i in 1:750) {
  bltest[i,]<-test[which(test$id==i),c(2,4,7)][1,]
}
```

```
bltest<-bltest[complete.cases(bltest),]

##Fit baseline logistic model to training set
bslogit<-glm(died~prothrombin+treatment,
                data=bltrain,family=binomial)

##Create function that returns sensitivity and specificity in AUC
sn.sp <- function(mat){
  sn <- mat[2,2]/sum(mat[2,])
  sp <- mat[1,1]/sum(mat[1,])
  return(unlist(list(sensitivity=sn, specificity=sp)))
}

##Predict by baseline logistic model
ytrainbl<-predict(bslogit,newdata=bltrain, type = "response")
ypredbl<-predict(bslogit,newdata=bltest, type = "response")
auc.trainbl <- roc(bltrain$died,ytrainbl)
auc.testbl <- roc(bltest$died,ypredbl)
snsp.trainbl<- cbind(auc.trainbl$sensitivities,
                        auc.trainbl$specificities)
indxbl<- which.max(apply(snsp.trainbl,1,min))
indxbl2<- which.max(apply(snsp.trainbl,1,sum))
cutoffbl<- auc.trainbl$thresholds[indxbl]
cutoffbl2<- auc.trainbl$thresholds[indxbl2]
sn.sp(table(bltest$died, as.numeric(ypredbl>cutoffbl)))
auc.testbl

##Brier Score
bsbl<-mean((ypredbl-bltest[,3])^2)
bsbl

##Fit LMM to the liver dataset
stage1<-lme(prothrombin~time+treatment, random=~1|id, data=train)

##Retrieve the fitted coefficients and random effects
for(i in 1:955){
  train[i,8]<-stage1$coefficients$random$id[which(
             rownames(stage1$coefficients$random$id)
             ==paste(train[i,1])),1]
}
alpha0<-stage1$coefficients$fixed[1]
alpha1<-stage1$coefficients$fixed[2]
alpha2<-stage1$coefficients$fixed[3]
s1<-as.numeric(VarCorr(stage1)[1,2])
s2<-as.numeric(VarCorr(stage1)[2,2])
```

```
##Keep time-fixed observations (1st observation only)
train2s<-data.frame(NA,NA,NA)
for (i in 1:486) {
  train2s[i,]<-train[which(train$id==i),c(4,8,7)][1,]
}
colnames(train2s)<-c("treatment","ric","died")
train2s<-train2s[complete.cases(train2s),]

##Fit stage2 logistic random intercept
stage2<-glm(died~treatment+ric, data=train2s,family=binomial)
beta0<-stage2$coefficients[1]
beta1<-stage2$coefficients[2]
gamma1<-stage2$coefficients[3]

##Calcualte ypred for test set
test$y<-(test$prothrombin-alpha0
            -alpha1*test$time-alpha2*test$treatment)
j4hold=lme(y~-1,random=~1|id,data=test)
U1j=j4hold$coefficients$random$id
test2s<-data.frame(NA,NA)
for (i in 1:488) {
  test2s[i,]<-test[which(test$id==i),c(4,7)][1,]
}
colnames(test2s)<-c("treatment","died")
test2s<-test2s[complete.cases(test2s),]
test2s$ric=U1j
ypred2s<-exp(beta0+beta1*test2s$treatment+gamma1*test2s$ric)/
  (1+exp(beta0+beta1*test2s$treatment+gamma1*test2s$ric))
ypred2s<-ypred2s[,1]

##Predict by 2-stage model
ytrain2s<-predict(stage2,newdata = train2s, type = "response")
auc.train2s <- roc(train2s$died,ytrain2s)
auc.test2s <- roc(test2s$died,ypred2s)
snsp.train2s<- cbind(auc.train2s$sensitivities,
                     auc.train2s$specificities)
indx2s<- which.max(apply(snsp.train2s,1,min))
indx2s2<- which.max(apply(snsp.train2s,1,sum))
cutoff2s<- auc.train2s$thresholds[indx2s]
cutoff2s2<- auc.train2s$thresholds[indx2s2]
sn.sp(table(test2s$died, as.numeric(ypred2s>cutoff2s)))
auc.test2s

##Brier Score
```

```
bs2s<-mean((ypred2s-test2s[,2])^2)
bs2s

##Train and test sets for feed-forward Neural Network
Xtrainfnn<-as.matrix(bltrain[,c(1:2)])
ytrainfnn<-to_categorical(bltrain[,3])
Xtestfnn<-as.matrix(bltest[,c(1:2)])
ytestfnn<-to_categorical(bltest[,3])

##Hyperparameter Grids for FNN
par<-list(
  neurons1=c(16,32),
  dropout1=c(0.2,0.3),
  activate1=c("relu","sigmoid"),
  neurons2=c(16,32),
  dropout2=c(0.2,0.3),
  activate2=c("relu","sigmoid"),
  neurons3=c(16,32),
  dropout3=c(0.2,0.3),
  activate3=c("relu","sigmoid"),
  activate4=c("sigmoid","softmax")
)

##Hyperparameter tuning, choosing the neural network
##with highest validation accuracy
liverfnn<-tuning_run("~/scratch/Data/liver/fnn.R", flags=par)
ls_runs(order="metric_val_loss",decreasing = F,
        runs_dir = "~/scratch/Data/liver/3lFNN/runs")[c(1:10),]
best_modelfnn<-load_model_hdf5("~/scratch/Data/liver/3lFNN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/runs/2021-03-09T22-41-33Z/modelfnn.h5")
##Predictions and performance
fnnpred<-best_modelfnn %>% predict(Xtestfnn)
fnntrain<- best_modelfnn %>% predict (Xtrainfnn)
auc.trainfnn <- roc(ytrainfnn[,2],fnntrain[,2])
auc.testfnn <- roc(ytestfnn[,2],fnnpred[,2])
snsp.trainfnn<- cbind(auc.trainfnn$sensitivities,
                      auc.trainfnn$specificities)
indxfnn<- which.max(apply(snsp.trainfnn,1,min))
indxfnn2<- which.max(apply(snsp.trainfnn,1,sum))
cutofffnn<- auc.trainfnn$thresholds[indxfnn]
cutofffnn2<- auc.trainfnn$thresholds[indxfnn2]
sn.sp(table(ytestfnn[,2], as.numeric(fnnpred[,2]>cutofffnn)))
auc.testfnn

##Brier Score
```

```
bsfnn<-mean((fnnpred[,2]-ytestfnn[,2])^2)
bsfnn

##Train and test set for long short-term memory
##Timestep padding
timestep<-c()
for(i in 1:488){
  timestep<-c(timestep,sum(liver$id==paste(i)))
}
maxtimestep<-max(timestep)
padding<-maxtimestep-timestep
padding<-padding[which(padding!=6)]
pad<-as.data.frame(matrix(-1,nrow=441,ncol=8))
pad<-cbind(pad,padding)
pad[,1]<-unique(liver$id)
pad1<-as.data.frame(lapply(pad,rep,pad$padding))
colnames(pad1)=c(colnames(liver),"padding")
liver<-rbind(liver,pad1[,-9])
liver<-liver[order(liver$time),]
liver<-liver[order(liver$id),]
liver<-liver[,-8]

Xtrainrnn<-array(as.matrix(liver[which(
                    liver$id %in% training.index),c(2:4)]),
                    dim = c(309,6,3))
ytrainrnn<-ytrainfnn
Xtestrnn<-array(as.matrix(liver[-which(
                    liver$id %in% training.index),c(2:4)]),
                    dim = c(132,6,3))
ytestrnn<-ytestfnn

##Hyperparameter grid for lstm
par2<-list(
  neurons1=c(8,16,32),
  dropout1=c(0.2,0.3,0.4),
  activate1=c("relu","sigmoid","softmax")
)

##Hyperparameter tuning, choosing the neural network
##with the highest validation accuracy
liverrnn<-tuning_run("~/scratch/Data/liver/rnn.R", flags=par2)
ls_runs(order="metric_val_loss",decreasing = F,
        runs_dir = "~/scratch/Data/liver/LSTM/runs")[c(1:10),]
best_modelrnn<-load_model_hdf5("~/scratch/Data/liver/LSTM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/runs/2021-03-09T18-19-54Z/modelrnn.h5")
```

```
##Predictions and performance
rnnpred<-best_modelrnn %>% predict(Xtestrnn)
rnntrain<- best_modelrnn %>% predict (Xtrainrnn)
auc.trainrnn <- roc(ytrainrnn[,2],rnntrain[,2])
auc.testrnn <- roc(ytestrnn[,2],rnnpred[,2])
snsp.trainrnn<- cbind(auc.trainrnn$sensitivities,
                      auc.trainrnn$specificities)
indxrnn<- which.max(apply(snsp.trainrnn,1,min))
indxrnn2<- which.max(apply(snsp.trainrnn,1,sum))
cutoffrnn<- auc.trainrnn$thresholds[indxrnn]
cutoffrnn2<- auc.trainrnn$thresholds[indxrnn2]
sn.sp(table(ytestrnn[,2], as.numeric(rnnpred[,2]>cutoffrnn)))
auc.testrnn

##Brier Score
bsrnn<-mean((rnnpred[,2]-ytestrnn[,2])^2)
bsrnn

#####~/scratch/Data/liver/fnn.R#####

FLAGS<-flags(
  flag_integer("neurons1", 32),
  flag_numeric("dropout1", 0.2),
  flag_string("activate1","relu"),
  flag_integer("neurons2", 32),
  flag_numeric("dropout2", 0.2),
  flag_string("activate2","relu"),
  flag_integer("neurons3", 32),
  flag_numeric("dropout3", 0.2),
  flag_string("activate3","relu"),
  flag_string("activate4","relu")
)

build_model<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_dense(units =FLAGS$neurons1,
                activation = FLAGS$activate1,
                input_shape = c(2)) %>%
    layer_dropout(FLAGS$dropout1) %>%
    layer_dense(units = FLAGS$neurons2,
                activation = FLAGS$activate2) %>%
    layer_dropout(FLAGS$dropout2) %>%
    layer_dense(units = FLAGS$neurons3,
                activation = FLAGS$activate3) %>%
```

```
    layer_dropout(FLAGS$dropout3) %>%
    layer_dense(units = 2, activation = FLAGS$activate4)

  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

model<- build_model()

history <- model %>% fit(
    Xtrainfnn, ytrainfnn,
    epochs = 200, batch_size = 309,
  validation_split = 0.2
)

plot(history)

score<- model %>% evaluate(Xtestfnn, ytestfnn,verbose = 0)

save_model_hdf5(model, "modelfnn.h5")

model

cat("Test loss:", getElement(score,"loss"), "\n")
cat("Test accuray:", getElement(score,"accuracy"), "\n")

#####~/scratch/Data/liver/rnn.R#####

FLAGS<-flags(
  flag_integer("neurons1", 32),
  flag_numeric("dropout1", 0.2),
  flag_string("activate1","relu")
)

build_model<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_masking(mask_value = -1,input_shape=c(6,3))%>%
    layer_lstm(units = FLAGS$neurons1, input_shape=c(6,3))%>%
    layer_dropout(FLAGS$dropout1)%>%
    layer_dense(units = 2, activation=FLAGS$activate1)
```

```
  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

model<- build_model()

history <- model %>% fit(
  Xtrainrnn, ytrainrnn,
  epochs = 500, batch_size = 309,
  validation_split = 0.2
)

plot(history)

score<- model %>% evaluate(Xtestrnn, ytestrnn,verbose = 0)

save_model_hdf5(model, "modelrnn.h5")

model

cat("Test loss:", getElement(score,"loss"), "\n")
cat("Test accuray:", getElement(score,"accuracy"), "\n")

#####################################
#########PBC2 Data Analysis##########
#####################################

#######SHARCNET Job Submission#######

#!/bin/bash
#SBATCH --time=20:00:00
#SBATCH --account=def-jhorrock
#SBATCH --job-name=pbc2
#SBATCH --ntasks=32
#SBATCH --mem-per-cpu=2GB
#SBATCH --output=slurm-%A_%a.out
#SBATCH --mail-user=zchen23@uoguelph.ca
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
```

```
#SBATCH --mail-type=REQUEUE
#SBATCH --mail-type=ALL

module load nixpkgs/16.09 gcc/8.3.0 r/4.0.0 python/3.6
source tensorflow/bin/activate
Rscript ~/scratch/Data/pbc2/pbc2.R

#####################################

#####~/scratch/Data/pbc2/pbc2.R######

##Working directory, packages and dataset
setwd("~/scratch/Data/pbc2")
library(keras)
use_session_with_seed(123)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(pROC)
library(JM)
library(tensorflow)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(tfruns)
data(pbc2)
set.seed(123)
pbc2<-pbc2[,-13]
pbc2<-pbc2[complete.cases(pbc2),]

##Died=0 if survived 5 year, =1 if died within 5 year,
##=2 if censored or transplanted
for (i in 1:1863) {
  if (pbc2[i,3] == "dead" & pbc2[i,2]<=5) pbc2[i,19]<-1
  else if (pbc2[i,2] >5) pbc2[i,19]<-0
  else pbc2[i,19]<-2
}

##Eliminate the censored data and
##the measurement time later than 5 year
pbc2<-pbc2[which(pbc2$status2<2&pbc2$year<=5),]
pbc2$drug<- ifelse(pbc2$drug=="placebo", 0, 1)
pbc2$sex<-ifelse(pbc2$sex=="female", 1, 0)
pbc2$ascites<-ifelse(pbc2$ascites=="Yes", 1, 0)
pbc2$hepatomegaly<-ifelse(pbc2$hepatomegaly=="Yes", 1, 0)
pbc2$spiders<-ifelse(pbc2$spiders=="Yes", 1, 0)
pbc2$edema<-ifelse(pbc2$edema=="No edema", 0, 1)

##Generate training IDs
```

```
training.index <- sample(unique(pbc2$id),203, replace = FALSE)

##Training and test sets
train<-pbc2[which(pbc2$id %in% training.index),]
test<-pbc2[-which(pbc2$id %in% training.index),]

##Logistic regression with first observation
bltrain<-data.frame(NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA)
colnames(bltrain)<-c(colnames(pbc2)[4:19])
bltest<-data.frame(NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA)
colnames(bltest)<-c(colnames(pbc2)[4:19])
for (i in 1:905) {
  bltrain[i,]<-train[which(train$id==i),c(4:19)][1,]
}
bltrain<-bltrain[complete.cases(bltrain),]
for (i in 1:399) {
  bltest[i,]<-test[which(test$id==i),c(4:19)][1,]
}
bltest<-bltest[complete.cases(bltest),]

#options("scipen"=100, "digits"=7)
##Fit baseline logistic model to training set
bslogit<-glm(status2~., data=bltrain,family=binomial)

##Backward AIC model selection
stepAIC(bslogit, direction = "backward")
bslogitAIC<-glm(status2~age+edema+serBilir
                +platelets+histologic,data=bltrain,family=binomial)

##Create function that returns sensitivity and specificity in AUC
sn.sp <- function(mat){
  sn <- mat[2,2]/sum(mat[2,])
  sp <- mat[1,1]/sum(mat[1,])
  return(unlist(list(sensitivity=sn, specificity=sp)))
}

##Predict by baseline logistic model
ytrainbl<-predict(bslogit,newdata=bltrain, type = "response")
ypredbl<-predict(bslogit,newdata=bltest, type = "response")
auc.trainbl <- roc(bltrain$status2,ytrainbl)
auc.testbl <- roc(bltest$status2,ypredbl)
snsp.trainbl<- cbind(auc.trainbl$sensitivities,
                     auc.trainbl$specificities)
indxbl<- which.max(apply(snsp.trainbl,1,min))
indxbl2<- which.max(apply(snsp.trainbl,1,sum))
```

```
cutoffbl<- auc.trainbl$thresholds[indxbl]
cutoffbl2<- auc.trainbl$thresholds[indxbl2]
sn.sp(table(bltest$status2, as.numeric(ypredbl>cutoffbl)))
auc.testbl
##Brier Score
bsbl<-mean((ypredbl-bltest[,16])^2)
bsbl

##Predict by AIC selected baseline logistic model
ytrainblAIC<-predict(bslogitAIC,newdata=bltrain, type = "response")
ypredblAIC<-predict(bslogitAIC,newdata=bltest, type = "response")
auc.trainblAIC <- roc(bltrain$status2,ytrainblAIC)
auc.testblAIC <- roc(bltest$status2,ypredblAIC)
snsp.trainblAIC<- cbind(auc.trainblAIC$sensitivities,
                        auc.trainblAIC$specificities)
indxblAIC<- which.max(apply(snsp.trainblAIC,1,min))
indxblAIC2<- which.max(apply(snsp.trainblAIC,1,sum))
cutoffblAIC<- auc.trainblAIC$thresholds[indxblAIC]
cutoffblAIC2<- auc.trainblAIC$thresholds[indxblAIC2]
sn.sp(table(bltest$status2, as.numeric(ypredblAIC>cutoffblAIC)))
auc.testblAIC
##Brier Score
bsblAIC<-mean((ypredblAIC-bltest[,16])^2)
bsblAIC

##Fit LMM to the pbc2 dataset
stage1<-lme(prothrombin~drug+age+sex+year+ascites+hepatomegaly
            +spiders+edema+serBilir+albumin+alkaline+SGOT+platelets
            +histologic, random=~1|id, data=train,method ="ML")

##AIC model selection
stepAIC(stage1,direction="backward")
stage1AIC<-lme(prothrombin~sex+ascites+hepatomegaly+spiders
               +edema+serBilir+albumin+alkaline+platelets,
               random=~1|id, data=train,method ="ML")

##Retrieve the fitted coefficients and random effects
for(i in 1:905){
  train$ric[i]<-stage1$coefficients$random$id[
    which(rownames(stage1$coefficients$random$id)
          ==paste(train[i,1])),1]
}
for(i in 1:905){
  train$ricAIC[i]<-stage1AIC$coefficients$random$id[
    which(rownames(stage1AIC$coefficients$random$id)
```

```
          ==paste(train[i,1])),1]
}
alpha0<-stage1$coefficients$fixed[1]
alpha1<-stage1$coefficients$fixed[2]
alpha2<-stage1$coefficients$fixed[3]
alpha3<-stage1$coefficients$fixed[4]
alpha4<-stage1$coefficients$fixed[5]
alpha5<-stage1$coefficients$fixed[6]
alpha6<-stage1$coefficients$fixed[7]
alpha7<-stage1$coefficients$fixed[8]
alpha8<-stage1$coefficients$fixed[9]
alpha9<-stage1$coefficients$fixed[10]
alpha10<-stage1$coefficients$fixed[11]
alpha11<-stage1$coefficients$fixed[12]
alpha12<-stage1$coefficients$fixed[13]
alpha13<-stage1$coefficients$fixed[14]
alpha14<-stage1$coefficients$fixed[15]
##AIC
alpha0AIC<-stage1AIC$coefficients$fixed[1]
alpha1AIC<-stage1AIC$coefficients$fixed[2]
alpha2AIC<-stage1AIC$coefficients$fixed[3]
alpha3AIC<-stage1AIC$coefficients$fixed[4]
alpha4AIC<-stage1AIC$coefficients$fixed[5]
alpha5AIC<-stage1AIC$coefficients$fixed[6]
alpha6AIC<-stage1AIC$coefficients$fixed[7]
alpha7AIC<-stage1AIC$coefficients$fixed[8]
alpha8AIC<-stage1AIC$coefficients$fixed[9]
alpha9AIC<-stage1AIC$coefficients$fixed[10]

##Keep time-fixed observations (1st observation only)
train2s<-data.frame(NA,NA,NA,NA,NA,NA)
for (i in 1:306) {
  train2s[i,]<-train[which(train$id==i),c(4,5,6,19,20,21)][1,]
}
colnames(train2s)<-c("drug","age","sex","status2","ric","ricAIC")
train2s<-train2s[complete.cases(train2s),]

##Fit stage2 logistic with estimated random intercept
stage2<-glm(status2~drug+age+sex+ric, data=train2s,family=binomial)
beta0<-stage2$coefficients[1]
beta1<-stage2$coefficients[2]
beta2<-stage2$coefficients[3]
beta3<-stage2$coefficients[4]
gamma1<-stage2$coefficients[5]
```

```
##Fit stage2 logistic with AIC stage 1's estimated random intercept
stage22<-glm(status2~drug+age+sex+ricAIC,
                data=train2s,family=binomial)
##AIC for stage 2
stepAIC(stage22,direction = "backward")
stage2AIC<-glm(status2~age+ricAIC, data=train2s,family=binomial)
beta0AIC<-stage2AIC$coefficients[1]
beta1AIC<-stage2AIC$coefficients[2]
gamma1AIC<-stage2AIC$coefficients[3]

##Calcualte ypred for test set for both model
test$y<-(test$prothrombin-alpha0-alpha1*test$drug-alpha2*test$age
         -alpha3*test$sex-alpha4*test$year-alpha5*test$ascites
         -alpha6*test$hepatomegaly-alpha7*test$spiders
         -alpha8*test$edema-alpha9*test$serBilir
         -alpha10*test$albumin-alpha11*test$alkaline
         -alpha12*test$SGOT-alpha13*test$platelets
         -alpha14*test$histologic)

test$yAIC<-(test$prothrombin-alpha0AIC-alpha1AIC*test$sex
            -alpha2AIC*test$ascites-alpha3AIC*test$hepatomegaly
            -alpha4AIC*test$spiders-alpha5AIC*test$edema
            -alpha6AIC*test$serBilir-alpha7AIC*test$albumin
            -alpha8AIC*test$alkaline-alpha9AIC*test$platelets)

j4hold=lme(y~-1,random=~1|id,data=test)
j4holdAIC=lme(yAIC~-1,random=~1|id,data=test)

U1j=j4hold$coefficients$random$id
U1jAIC=j4holdAIC$coefficients$random$id
test2s<-data.frame(NA,NA,NA,NA)
for (i in 1:302) {
  test2s[i,]<-test[which(test$id==i),c(4,5,6,19)][1,]
}
colnames(test2s)<-c("drug","age","sex","status2")
test2s<-test2s[complete.cases(test2s),]
test2s$ric=U1j
test2s$ricAIC=U1jAIC
ypred2s<-exp(beta0+beta1*test2s$drug+beta2*test2s$age
                +beta3*test2s$sex+gamma1*test2s$ric)/
                (1+exp(beta0+beta1*test2s$drug+beta2*test2s$age
                +beta3*test2s$sex+gamma1*test2s$ric))
ypred2sAIC<-exp(beta0AIC+beta1AIC*test2s$age
                +gamma1AIC*test2s$ricAIC)/(1+exp(beta0AIC
                +beta1AIC*test2s$age+gamma1AIC*test2s$ricAIC))
```

```
ypred2s<-ypred2s[,1]
ypred2sAIC<-ypred2sAIC[,1]

##Predict by 2-stage model
ytrain2s<-predict(stage2,newdata = train2s, type = "response")
auc.train2s <- roc(train2s$status2,ytrain2s)
auc.test2s <- roc(test2s$status2,ypred2s)
snsp.train2s<- cbind(auc.train2s$sensitivities,
                     auc.train2s$specificities)
indx2s<- which.max(apply(snsp.train2s,1,min))
indx2s2<- which.max(apply(snsp.train2s,1,sum))
cutoff2s<- auc.train2s$thresholds[indx2s]
cutoff2s2<- auc.train2s$thresholds[indx2s2]
sn.sp(table(test2s$status2, as.numeric(ypred2s>cutoff2s)))
auc.test2s

##Brier Score 2-stage model
bs2s<-mean((ypred2s-test2s[,4])^2)
bs2s

##Predict by 2-stage model with AIC
ytrain2sAIC<-predict(stage2AIC,newdata = train2s, type = "response")
auc.train2sAIC <- roc(train2s$status2,ytrain2sAIC)
auc.test2sAIC <- roc(test2s$status2,ypred2sAIC)
snsp.train2sAIC<- cbind(auc.train2sAIC$sensitivities,
                     auc.train2sAIC$specificities)
indx2sAIC<- which.max(apply(snsp.train2sAIC,1,min))
indx2sAIC2<- which.max(apply(snsp.train2sAIC,1,sum))
cutoff2sAIC<- auc.train2sAIC$thresholds[indx2sAIC]
cutoff2sAIC2<- auc.train2sAIC$thresholds[indx2sAIC2]
sn.sp(table(test2s$status2, as.numeric(ypred2sAIC>cutoff2sAIC)))
auc.test2sAIC

##Brier Score AIC 2-stage model
bs2sAIC<-mean((ypred2sAIC-test2s[,4])^2)
bs2sAIC

##Train and test sets for feed-forward Neural Network
Xtrainfnn<-as.matrix(bltrain[,c(1:15)])
ytrainfnn<-to_categorical(bltrain[,16])
Xtestfnn<-as.matrix(bltest[,c(1:15)])
ytestfnn<-to_categorical(bltest[,16])

##Hyperparameter Grids for FNN
par<-list(
```

```
  neurons1=c(16,32),
  dropout1=c(0.2,0.3),
  activate1=c("relu","sigmoid"),
  neurons2=c(16,32),
  dropout2=c(0.2,0.3),
  activate2=c("relu","sigmoid"),
  neurons3=c(16,32),
  dropout3=c(0.2,0.3),
  activate3=c("relu","sigmoid"),
  activate4=c("sigmoid","softmax")
)

##Hyperparameter tuning, choosing the neural network with
##the highest validation accuracy
pbc2fnn<-tuning_run("~/scratch/Data/pbc2/fnn.R", flags=par)
ls_runs(order="metric_val_loss",decreasing = F,
        runs_dir = "~/scratch/Data/pbc2/3lFNN/runs")[c(1:10),]
best_modelfnn<-load_model_hdf5("~/scratch/Data/pbc2/3lFNN/
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣runs/2021-03-10T06-56-36Z/modelfnn.h5")

##Predictions and performance
fnnpred<-best_modelfnn %>% predict(Xtestfnn)
fnntrain<- best_modelfnn %>% predict (Xtrainfnn)
auc.trainfnn <- roc(ytrainfnn[,2],fnntrain[,2])
auc.testfnn <- roc(ytestfnn[,2],fnnpred[,2])
snsp.trainfnn<- cbind(auc.trainfnn$sensitivities,
                      auc.trainfnn$specificities)
indxfnn<- which.max(apply(snsp.trainfnn,1,min))
indxfnn2<- which.max(apply(snsp.trainfnn,1,sum))
cutofffnn<- auc.trainfnn$thresholds[indxfnn]
cutofffnn2<- auc.trainfnn$thresholds[indxfnn2]
sn.sp(table(ytestfnn[,2], as.numeric(fnnpred[,2]>cutofffnn)))
auc.testfnn

##Brier Score
bsfnn<-mean((fnnpred[,2]-ytestfnn[,2])^2)
bsfnn

##Train and test set for long short-term memory
##Timestep padding
timestep<-c()
for(i in 1:306){
  timestep<-c(timestep,sum(pbc2$id==paste(i)))
}
maxtimestep<-max(timestep)
```

```
padding<-maxtimestep-timestep
padding<-padding[which(padding!=7)]
pad<-as.data.frame(matrix(-1,nrow=290,ncol=19))
pad<-cbind(pad,padding)
pad[,1]<-unique(pbc2$id)
pad1<-as.data.frame(lapply(pad,rep,pad$padding))
colnames(pad1)=c(colnames(pbc2),"padding")
pbc2<-rbind(pbc2,pad1[,-20])
pbc2<-pbc2[order(pbc2$year),]
pbc2<-pbc2[order(pbc2$id),]

Xtrainrnn<-array(as.matrix(pbc2[
            which(pbc2$id %in% training.index),c(4:18)]),
            dim = c(203,7,15))
ytrainrnn<-ytrainfnn
Xtestrnn<-array(as.matrix(pbc2[
            -which(pbc2$id %in% training.index),c(4:18)]),
            dim = c(87,7,15))
ytestrnn<-ytestfnn

##Hyperparameter grid for lstm
par2<-list(
  neurons1=c(8,16,32),
  dropout1=c(0.2,0.3,0.4),
  activate1=c("relu","sigmoid","softmax")
)

##Hyperparameter tuning, choosing the neural network with
##the highest validation accuracy
pbc2rnn<-tuning_run("~/scratch/Data/pbc2/rnn.R", flags=par2)
ls_runs(order="metric_val_loss",decreasing = F,
        runs_dir = "~/scratch/Data/pbc2/LSTM/runs")[c(1:10),]
best_modelrnn<-load_model_hdf5("~/scratch/Data/pbc2/LSTM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/runs/2021-03-10T06-29-38Z/modelrnn.h5")
##Predictions and performance
rnnpred<-best_modelrnn %>% predict(Xtestrnn)
rnntrain<- best_modelrnn %>% predict (Xtrainrnn)
auc.trainrnn <- roc(ytrainrnn[,2],rnntrain[,2])
auc.testrnn <- roc(ytestrnn[,2],rnnpred[,2])
snsp.trainrnn<- cbind(auc.trainrnn$sensitivities,
                      auc.trainrnn$specificities)
indxrnn<- which.max(apply(snsp.trainrnn,1,min))
indxrnn2<- which.max(apply(snsp.trainrnn,1,sum))
cutoffrnn<- auc.trainrnn$thresholds[indxrnn]
cutoffrnn2<- auc.trainrnn$thresholds[indxrnn2]
```

```
sn.sp(table(ytestrnn[,2], as.numeric(rnnpred[,2]>cutoffrnn)))
auc.testrnn
##Brier Score
bsrnn<-mean((rnnpred[,2]-ytestrnn[,2])^2)
bsrnn

#####~/scratch/Data/pbc2/fnn.R#####

FLAGS<-flags(
  flag_integer("neurons1", 32),
  flag_numeric("dropout1", 0.2),
  flag_string("activate1","relu"),
  flag_integer("neurons2", 32),
  flag_numeric("dropout2", 0.2),
  flag_string("activate2","relu"),
  flag_integer("neurons3", 32),
  flag_numeric("dropout3", 0.2),
  flag_string("activate3","relu"),
  flag_string("activate4","relu")
)

build_model<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_dense(units =FLAGS$neurons1,
                activation = FLAGS$activate1,
                input_shape = c(15)) %>%
    layer_dropout(FLAGS$dropout1) %>%
    layer_dense(units = FLAGS$neurons2,
                activation = FLAGS$activate2) %>%
    layer_dropout(FLAGS$dropout2) %>%
    layer_dense(units = FLAGS$neurons3,
                activation = FLAGS$activate3) %>%
    layer_dropout(FLAGS$dropout3) %>%
    layer_dense(units = 2, activation = FLAGS$activate4)

  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

model<- build_model()
```

```
history <- model %>% fit(
  Xtrainfnn, ytrainfnn,
  epochs = 200, batch_size = 203,
  validation_split = 0.2
)

plot(history)

score<- model %>% evaluate(Xtestfnn, ytestfnn,verbose = 0)

save_model_hdf5(model, "modelfnn.h5")

model

cat("Test loss:", getElement(score,"loss"), "\n")
cat("Test accuray:", getElement(score,"accuracy"), "\n")

#####~/scratch/Data/pbc2/rnn.R#####

FLAGS<-flags(
  flag_integer("neurons1", 32),
  flag_numeric("dropout1", 0.2),
  flag_string("activate1","relu")
)

build_model<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_masking(mask_value = -1,input_shape=c(7,15))%>%
    layer_lstm(units = FLAGS$neurons1, input_shape=c(7,15))%>%
    layer_dropout(FLAGS$dropout1)%>%
    layer_dense(units = 2, activation=FLAGS$activate1)

  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

model<- build_model()

history <- model %>% fit(
```

```
  Xtrainrnn, ytrainrnn,
  epochs = 500, batch_size = 203,
  validation_split = 0.2
)

plot(history)

score<- model %>% evaluate(Xtestrnn, ytestrnn,verbose = 0)

save_model_hdf5(model, "modelrnn.h5")

model

cat("Test loss:", getElement(score,"loss"), "\n")
cat("Test accuray:", getElement(score,"accuracy"), "\n")

#####################################
#########Simulation Study############
#####################################

#######SHARCNET Job Submission#######

#!/bin/bash
#SBATCH --time=20:00:00
#SBATCH --account=def-jhorrock
#SBATCH --job-name=liversim
#SBATCH --ntasks=32
#SBATCH --mem-per-cpu=2GB
#SBATCH --output=slurm-%A_%a.out
#SBATCH --mail-user=zchen23@uoguelph.ca
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
#SBATCH --mail-type=REQUEUE
#SBATCH --mail-type=ALL

module load nixpkgs/16.09 gcc/8.3.0 r/4.0.0 python/3.6
source tensorflow/bin/activate
Rscript ~/scratch/Data/liversim.R

#####################################

######~/scratch/Data/liversim.R######

##Working directory, packages and dataset
```

```
setwd("~/scratch/Data")
library(keras)
use_session_with_seed(123)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(lattice)
library(joineR)
library(JM)
library(tensorflow)
use_virtualenv(Sys.getenv("VIRTUAL_ENV"))
library(pROC)
library(tfruns)
data(liver)
set.seed(123)
liver=data.frame(liver,NA,NA)
colnames(liver)=c("id","prothrombin","time","treatment",
                  "survival","cens","died","ric")

##Died=0 if survived 1 year, =1 if died within 1 year,
##=2 if censored
for (i in 1:2969) {
  if (liver[i,6] == 1&liver[i,5]<=1) liver[i,7]<-1
  else if (liver[i,5] >1) liver[i,7]<-0
  else liver[i,7]<-2
}
##Eliminate the censored data and the
##measurement time later than 1 year
liver<-liver[which(liver$died<2&liver$time<=1),]

##Fit LMM to the liver dataset
stage1<-lme(prothrombin~time+treatment, random=~1|id, data=liver)

##Retrieve the fitted coefficients and random effects
for(i in 1:1364){
  liver[i,8]<-stage1$coefficients$random$id[
               which(rownames(stage1$coefficients$random$id)
               ==paste(liver[i,1])),1]
}
alpha0<-stage1$coefficients$fixed[1]
alpha1<-stage1$coefficients$fixed[2]
alpha2<-stage1$coefficients$fixed[3]
s1<-as.numeric(VarCorr(stage1)[1,2])
s2<-as.numeric(VarCorr(stage1)[2,2])

##Keep time-fixed observations (1st observation only)
Liver<-data.frame(NA,NA,NA,NA,NA,NA,NA,NA)
```

```
for (i in 1:488) {
  Liver[i,]<-liver[which(liver$id==i),][1,]
}
liver<-Liver[complete.cases(Liver),]
rm(Liver)
colnames(liver)<-c("id","prothrombin","time","treatment",
                   "survival","cens","died","ric")

##Calculate the proportion of treatment
px=sum(liver$treatment)/sum(nrow(liver))

##Fit stage2 logistic random intercept
stage2<-glm(died~treatment+ric, data=liver,family=binomial)

##Retrieve the fitted coefficients
beta0<-stage2$coefficients[1]
beta1<-stage2$coefficients[2]
gamma1<-stage2$coefficients[3]

##Setting initial parameters
N=500
t=5
iter=1000
xi<-rep(rbinom(N,1,px),each=t)
tj<-rep(c(1:5)/(t+1),N)
id<-rep(c(1:N),each=t)
sensitbl=sensit2s=sensitfnn=sensitrnn=c()
specifbl=specif2s=speciffnn=specifrnn=c()
aucbl=auc2s=aucfnn=aucrnn=c()
bsbl=bs2s=bsfnn=bsrnn=c()

##Create function that returns sensitivity and specificity in AUC
sn.sp <- function(mat){
  sn <- mat[2,2]/sum(mat[2,])
  sp <- mat[1,1]/sum(mat[1,])
  return(unlist(list(sensitivity=sn, specificity=sp)))
}
##Simulation loop
for (j in 1:iter){

  ##Generate epsilon_i, U_i,calculate Wij,
  ##pi and deadi from liver data
  epsl_i<-rnorm(N*t,mean=0,sd=s2)
  U_i<-rep(rnorm(N, mean=0,sd=s1),each=t)
  Wij=alpha0+alpha1*tj+alpha2*xi+U_i+epsl_i
```

```
pi<-exp(beta0+beta1*xi+gamma1*U_i)/
  (1+exp(beta0+beta1*xi+gamma1*U_i))
diedi=c()

for(i in 1:N){
  diedi<-c(diedi,rep(rbinom(1,1,pi[5*(i-1)+1]),each=t))
}

##liversim dataframe
liversim<-data.frame(id,Wij,tj,xi,U_i,pi,diedi)
colnames(liversim)<-c("id","prothrombin","time",
                      "treatment","ric","pi","died")

##Generate training IDs
training.index <- sample.int(500,350, replace = FALSE)
##Training and test sets
trainsim<-liversim[which(liversim$id %in% training.index),]
testsim<-liversim[-which(liversim$id %in% training.index),]
##Logistic regression with first observation
bltrainsim<-data.frame(NA,NA,NA)
colnames(bltrainsim)<-c("prothrombin","treatment","died")
bltestsim<-data.frame(NA,NA,NA)
colnames(bltestsim)<-c("prothrombin","treatment","died")
for (i in 1:1750) {
  bltrainsim[i,]<-trainsim[which(trainsim$id==i),c(2,4,7)][1,]
}
bltrainsim<-bltrainsim[complete.cases(bltrainsim),]
for (i in 1:750) {
  bltestsim[i,]<-testsim[which(testsim$id==i),c(2,4,7)][1,]
}
bltestsim<-bltestsim[complete.cases(bltestsim),]

##Fit baseline logistic model to training set
bslogitsim<-glm(died~prothrombin+treatment,
                data=bltrainsim,family=binomial)

##Fit the LMM to training set
stage1sim<-lme(prothrombin~time+factor(treatment),
                 random=~1|id, data=trainsim)
alpha0sim<-stage1sim$coefficients$fixed[1]
alpha1sim<-stage1sim$coefficients$fixed[2]
alpha2sim<-stage1sim$coefficients$fixed[3]

##Retrieve U_ihat
for(i in 1:1750){
```

```
  trainsim[i,5]<-stage1sim$coefficients$random$id[
                  which(rownames(stage1sim$coefficients$random$id)
                  ==paste(trainsim[i,1])),1]
}
trainsimstage2<-trainsim[round(trainsim$time,2)== 0.17,]

##Fit the logistic regression to training set
stage2sim<-glm(died~treatment+ric,
                  data=trainsimstage2,family=binomial)
beta0sim<-stage2sim$coefficients[1]
beta1sim<-stage2sim$coefficients[2]
gamma1sim<-stage2sim$coefficients[3]

##Calcualte ypred for test set
testsim$y<-(testsim$prothrombin-alpha0sim
              -alpha1sim*testsim$time-alpha2sim*testsim$treatment)
j4hold=lme(y~-1,random=~1|id,data=testsim)
U1j=j4hold$coefficients$random$id
testsimstage2<-testsim[round(testsim$time,2)== 0.17,]
testsimstage2$ric=U1j
ypred2s<-exp(beta0sim+beta1sim*testsimstage2$treatment
              +gamma1sim*testsimstage2$ric)/
              (1+exp(beta0sim+beta1sim*testsimstage2$treatment
              +gamma1sim*testsimstage2$ric))
ypred2s<-ypred2s[,1]
##Train 3 hidden layer FNN with random search
Xtrainsim<-as.matrix(bltrainsim[,c(1,2)])
ytrainsim<-to_categorical(bltrainsim[,3])
Xtestsim<-as.matrix(bltestsim[,c(1,2)])
ytestsim<-to_categorical(bltestsim[,3])

build_modelfnn<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 32, activation = "relu",
                  input_shape = c(2)) %>%
    layer_dropout(0.3) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dropout(0.2) %>%
    layer_dense(units = 32, activation = "sigmoid") %>%
    layer_dropout(0.3) %>%
    layer_dense(units = 2, activation = "softmax")

  model %>% compile(
    loss = "binary_crossentropy",
```

```
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

modelfnn<- build_modelfnn()

history <- modelfnn %>% fit(
  Xtrainsim, ytrainsim,
  epochs = 200, batch_size = 350,
  validation_split = 0.2
)

##Train 1 layer RNN with random search
Xtrainsim2<-array(as.matrix(liversim[
                  which(liversim$id %in% training.index),c(2,4)]),
                  dim = c(350,5,3))
ytrainsim2<-to_categorical(liversim[
                  which(liversim$id %in% training.index),7])
ytrainsim2<-ytrainsim2[seq(1, 1750, 5),]
Xtestsim2<-array(as.matrix(liversim[
              -which(liversim$id %in% training.index),c(2,4)]),
              dim = c(150,5,3))
ytestsim2<-to_categorical(liversim[
                  -which(liversim$id %in% training.index),7])
ytestsim2<-ytestsim2[seq(1, 750, 5),]

build_modelrnn<-function(){
  model <- keras_model_sequential()
  model %>%
    layer_lstm(units =  32, input_shape = c(5,3)) %>%
    layer_dropout(0.2)%>%
    layer_dense(units = 2, activation = "softmax")

  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = c("accuracy")
  )
  model
}

modelrnn<- build_modelrnn()
```

```
history <- modelrnn %>% fit(
  Xtrainsim2, ytrainsim2,
  epochs = 500, batch_size = 350,
  validation_split = 0.2
)

##Prediction and performance metrics records
ytrainbl<-predict(bslogitsim,bltrainsim, type = "response")
ypredbl<-predict(bslogitsim,bltestsim, type = "response")
ytrain2s<-predict(stage2sim,trainsimstage2, type = "response")
ytrainfnn<-modelfnn %>% predict(Xtrainsim)
ypredfnn<-modelfnn %>% predict(Xtestsim)
ytrainrnn<-modelrnn %>% predict(Xtrainsim2)
ypredrnn<-modelrnn %>% predict(Xtestsim2)
auc.trainbl <- roc(ytrainsim[,2],ytrainbl)
auc.train2s <- roc(ytrainsim[,2],ytrain2s)
auc.trainfnn <- roc(ytrainsim[,2],ytrainfnn[,2])
auc.trainrnn <- roc(ytrainsim2[,2],ytrainrnn[,2])
auc.testbl <- roc(ytestsim[,2],ypredbl)
auc.test2s <- roc(ytestsim[,2],ypred2s)
auc.testfnn <- roc(ytestsim[,2],ypredfnn[,2])
auc.testrnn <- roc(ytestsim2[,2],ypredrnn[,2])
snsp.trainbl<- cbind(auc.trainbl$sensitivities,
                     auc.trainbl$specificities)
indxbl<- which.max(apply(snsp.trainbl,1,min))
cutoffbl<- auc.trainbl$thresholds[indxbl]
snsp.train2s<- cbind(auc.train2s$sensitivities,
                     auc.train2s$specificities)
indx2s<- which.max(apply(snsp.train2s,1,min))
cutoff2s<- auc.train2s$thresholds[indx2s]
snsp.trainfnn<- cbind(auc.trainfnn$sensitivities,
                     auc.trainfnn$specificities)
indxfnn<- which.max(apply(snsp.trainfnn,1,min))
cutofffnn<- auc.trainfnn$thresholds[indxfnn]
snsp.trainrnn<- cbind(auc.trainrnn$sensitivities,
                     auc.trainrnn$specificities)
indxrnn<- which.max(apply(snsp.trainrnn,1,min))
cutoffrnn<- auc.trainrnn$thresholds[indxrnn]
sensitbl<-c(sensitbl,sn.sp(table(ytestsim[,2],
                          as.numeric(ypredbl>cutoffbl)))[1])
specifbl<-c(specifbl,sn.sp(table(ytestsim[,2],
                          as.numeric(ypredbl>cutoffbl)))[2])
sensit2s<-c(sensit2s,sn.sp(table(ytestsim[,2],
                          as.numeric(ypred2s>cutoff2s)))[1])
specif2s<-c(specif2s,sn.sp(table(ytestsim[,2],
```

```
                              as.numeric(ypred2s>cutoff2s)))[2])
  sensitfnn<-c(sensitfnn,sn.sp(table(ytestsim[,2],
                              as.numeric(ypredfnn[,2]>cutofffnn)))[1])
  speciffnn<-c(speciffnn,sn.sp(table(ytestsim[,2],
                              as.numeric(ypredfnn[,2]>cutofffnn)))[2])
  sensitrnn<-c(sensitrnn,sn.sp(table(ytestsim2[,2],
                              as.numeric(ypredrnn[,2]>cutoffrnn)))[1])
  specifrnn<-c(specifrnn,sn.sp(table(ytestsim2[,2],
                              as.numeric(ypredrnn[,2]>cutoffrnn)))[2])
  aucbl<-c(aucbl,auc.testbl$auc)
  auc2s<-c(auc2s,auc.test2s$auc)
  aucfnn<-c(aucfnn,auc.testfnn$auc)
  aucrnn<-c(aucrnn,auc.testrnn$auc)
  bsbl<-c(bsbl,mean((ypredbl-ytestsim[,2])^2))
  bs2s<-c(bs2s,mean((ypred2s-ytestsim[,2])^2))
  bsfnn<-c(bsfnn,mean((ypredfnn[,2]-ytestsim[,2])^2))
  bsrnn<-c(bsrnn,mean((ypredrnn[,2]-ytestsim2[,2])^2))
}
##Mean and SD of performance metrics
mean(aucbl)
mean(auc2s)
mean(aucfnn)
mean(aucrnn)
mean(sensitbl)
mean(sensit2s)
mean(sensitfnn)
mean(sensitrnn)
mean(specifbl)
mean(specif2s)
mean(speciffnn)
mean(specifrnn)
mean(bsbl)
mean(bs2s)
mean(bsfnn)
mean(bsrnn)

sd(aucbl)
sd(auc2s)
sd(aucfnn)
sd(aucrnn)
sd(sensitbl)
sd(sensit2s)
sd(sensitfnn)
sd(sensitrnn)
sd(specifbl)
```

```
sd(specif2s)
sd(speciffnn)
sd(specifrnn)
sd(bsbl)
sd(bs2s)
sd(bsfnn)
sd(bsrnn)
```