An R TensorFlow Codebook

Navarun Jain

This Codebook explores using TensorFlow in R through the Keras API to build and train neural networks. This Codebook contains the script used to build models on the `dataCar` dataset from R's `insuranceData` package, and is an accompaniment to my presentation on AI in Insurance Pricing, given at GIRO 2018. Hope you find this useful!

What you need to access and run this Codebook

1. R and RStudio - If you do not have these installed you can find them here: *R - https://cran.r-project.org/* RStudio - https://www.rstudio.com/products/rstudio/download/

2. R packages - The following are required: `insuranceData`, `data.table`, `MASS`, `boot`, `tweedie`, `statmod`. `caret`, `dummies`, `keras` and `dplyr`. If any of these packages are not installed on your system then you can install them using the command `install_packages("package")`.

Understanding Keras & TF

To learn more about Keras and it's different implementations please visit https://keras.io/. The R interface to Keras can be found here - https://keras.rstudio.com/.

To learn more about TensorFlow, please visit https://www.tensorflow.org/.

Before using Keras in R, it is necessary to install TensorFlow on your system. The recommended way is to do it through Anaconda on Windows or native pip on Mac OS. Detailed instructions can be found here - https://www.tensorflow.org/install/.

Once TensorFlow has been installed, install the Keras package and load it using `library(keras)`. After that, run `install_keras()` (if you are running TensorFlow on a GPU then run `install_keras(tensorflow = "gpu")`). This Codebook assumes that TF is being run on a CPU.

PLEASE NOTE Only run `install_keras()` once. After you run it please comment it out, otherwise it'll reboot your R session everytime you run any bit of code.

The Code

Please Note: By default, this code will not run when you try to knit this workbook. To make sure the code runs, please set `eval=TRUE` in all chunks you wish to run.

Dataset

```r
data("dataCar")

dataCar$veh_body <- as.factor(dataCar$veh_body)
dataCar$veh_age <- as.factor(dataCar$veh_age)
dataCar$gender <- as.factor(dataCar$gender)
dataCar$area <- as.factor(dataCar$area)
dataCar$agecat <- as.factor(dataCar$agecat)

clm_0_perc <- mean(dataCar$claimcst0 == 0)*100 #Percentage of records with 0 claims
clm_0_perc

hist(dataCar$claimcst0, breaks = 200, xlab = "Claim Amounts", ylab = "Count of Observations", main = "Histogram of Claim Costs", col = "blue")
hist(dataCar[dataCar$claimcst0 > 0, ]$claimcst0, breaks = 200, xlab = "Claim Amounts", ylab = "Count of Observations", main = "Histogram of Non-Zero Claim Costs", col = "blue")
hist(log(dataCar$claimcst0), xlab = "Log of Claim Amounts", ylab = "Count of Observations", main = "Histogram of Log Claim Costs", breaks = 200, col = "blue")

dataCar_active <- dataCar[, -c(3, 11)] #Removing unnecessary variables

#Building Aggregated Dataset
groupvars <- c("veh_value", "veh_body", "veh_age", "gender", "area", "agecat")
datavars <- c("exposure", "numclaims", "claimcst0")
dataCar_active <- as.data.table(dataCar_active)
dataCar_active <- dataCar_active[, lapply(.SD, sum), by = groupvars, .SDcols = datavars]
dataCar_active <- as.data.frame(dataCar_active)

#Severity
dataCar_active$severity <- c()
for(i in 1:nrow(dataCar_active)) {
  if(dataCar_active$numclaims[i] == 0) {
    dataCar_active$severity[i] <- 0
  } else {
    dataCar_active$severity[i] <- as.numeric(dataCar_active$claimcst0[i]) / as.numeric(dataCar_active$numclaims[i])
  }
}

#Train-Test Splits
set.seed(1002)
ind <- sample(1:nrow(dataCar_active), round(0.2*nrow(dataCar_active)))
train_data <- dataCar_active[-ind, ] #80% for training and validation
test_data <- dataCar_active[ind, ] #20% for test
```

Poisson-Gamma GLM

Poisson Model for Frequency

```
targetvar <- c("numclaims")
predictors <- c("veh_value", "veh_body", "veh_age", "gender", "area",
"agecat")
predictors <- paste(predictors, collapse = "+")
fre_glm_formula <- as.formula(paste(targetvar,"~",predictors, collapse =
"+"))

freq_glm <- glm(fre_glm_formula, data = train_data, family = poisson(link =
"log"), offset = log(exposure))
summary(freq_glm)
```

Gamma Model for Severity

```
targetvar <- c("severity")
predictors <- c("veh_value", "veh_body", "veh_age", "gender", "area",
"agecat")
predictors <- paste(predictors, collapse = "+")
sev_glm_formula <- as.formula(paste(targetvar,"~",predictors, collapse =
"+"))

sev_glm <- glm(sev_glm_formula, data = train_data[train_data$severity > 0, ],
family = Gamma(link = "log"))
summary(sev_glm)
```

5-fold Cross Validation

```
set.seed(5)
folds <- createFolds(dataCar_active$claimcst0, k = 5)

pg_glm_mse <- c()
for (i in 1:length(folds)) {
  train <- dataCar_active[-folds[[i]], ]
  test <- dataCar_active[folds[[i]], ]
  freq_mod <- glm(fre_glm_formula, data = train, family = poisson(link =
"log"), offset = log(exposure))
  sev_mod <-  glm(sev_glm_formula, data = train[train$severity > 0, ], family
= Gamma(link = "log"))
  fre_preds <- predict(freq_mod, test, type = "response")
  sev_preds <- predict(sev_mod, test, type = "response")
  predictions <- fre_preds * sev_preds
  pg_glm_mse[i] <- sum((predictions - test$claimcst0)^2)/nrow(test)
}

pg_glm_cv_mse <- mean(pg_glm_mse) #This is the 5-fold CV error
pg_glm_cv_mse
```

Tweedie GLM

```
targetvar <- c("claimcst0")
predictors <- c("veh_value", "veh_body", "veh_age", "gender", "area",
"agecat")
predictors <- paste(predictors, collapse = "+")
tweedie_glm_formula <- as.formula(paste(targetvar,"~",predictors, collapse =
"+"))
```

Tuning for variance power parameter

```
p_values <- seq(1.1, 1.7, by = 0.1)

p_tuning <- tweedie.profile(formula = tweedie_glm_formula, p.vec = p_values,
data = train_data, do.plot = TRUE, offset = log(exposure))
abline(v = p_tuning$p.max, col = "blue")

#Log-likehilood estimates for each chosed value of p
logliks <- data.frame(p_values, p_tuning$L, p_tuning$phi)
colnames(logliks) <- c("Variance power (p)", "Log-likelihood at p", "phi")
logliks

optimal_p <- p_tuning$p.max
optimal_p
```

Fitting Tweedie GLM

```
tweedie_model <- glm(formula = tweedie_glm_formula, data = train_data, family
= tweedie(var.power = optimal_p, link.power = 0), offset = log(exposure))
summary(tweedie_model)
```

5-fold Cross-validation of Tweedie GLM

```
set.seed(5)
folds <- createFolds(dataCar_active$claimcst0, k = 5)

tweedie_glm_mse <- c()
for (i in 1:length(folds)) {
  train <- dataCar_active[-folds[[i]], ]
  test <- dataCar_active[folds[[i]], ]
  model <- glm(formula = tweedie_glm_formula, data = train, family =
tweedie(var.power = optimal_p, link.power = 0), offset = log(exposure))
  predictions <- predict(model, test, type = "response")
  tweedie_glm_mse[i] <- sum((predictions - test$claimcst0)^2)/nrow(test)
}

tweedie_glm_cv_mse <- mean(tweedie_glm_mse)
tweedie_glm_cv_mse
```

Artificial Neural Networks

Before training neural networks, the data has to be processed as follows:

*Step 1: Since Neural Networks do not recognize character/factor variables, the data needs to be dummified, so that each level of a categorical variable becomes a column in the data, and every instance of a level is coded as a 1 or 0. This process is known as one-hot-encoding. Thus, while in the data, there were only 7 rating factors (exposure was added here), one-hot-encoding raised this number to 33. Thus, the input layer consists of 33 neurons.

*Step 2: In order to increase training efficiency and accuracy, the data needs to be scaled. It can be normalized, or scaled such that all values are between 1 and 0. In this example, the latter method has been used.

Data Preparation

```
targetvar <- c("claimcst0")
predictors <- c("veh_value", "veh_body", "veh_age", "gender", "area",
"agecat", "exposure")

#Creating Dummy Indicator Variables
factors <- c()
for (i in 1:length(predictors)) {
  if (class(dataCar_active[, predictors[i]]) == "factor") {
    factors[i] <- predictors[i]
  } else {
    i <- i + 1
  }
}

factors <- factors[!factors %in% NA]

#Dummifying Datasets
dataCar_active_nn <- dummy.data.frame(dataCar_active, names = factors)
dataCar_active_nn <- as.data.frame(lapply(dataCar_active_nn, as.numeric))
train_data_nn <- dummy.data.frame(train_data, names = factors)
train_data_nn <- as.data.frame(lapply(train_data_nn, as.numeric))
test_data_nn <- dummy.data.frame(test_data, names = factors)
test_data_nn <- as.data.frame(lapply(test_data_nn, as.numeric))

allvars <- colnames(dataCar_active_nn)
predictors <- allvars[!allvars %in% c(targetvar, "numclaims", "severity",
"claimcst0")]

#Scaling Training and Test Data
max <- apply(dataCar_active_nn, 2, max)
min <- apply(dataCar_active_nn, 2, min)

train_data_nn <- as.data.frame(scale(train_data_nn, center = min, scale =
(max - min)))
test_data_nn <- as.data.frame(scale(test_data_nn, center = min, scale = (max
- min)))
```

```
#Data Prep
x_train <- as.matrix(train_data_nn[predictors])
y_train <- as.matrix(train_data_nn[targetvar])

x_test <- as.matrix(test_data_nn[predictors])
y_test <- as.matrix(test_data_nn[targetvar])
```

Following is the code used to build and train each neural network that was tested. This code, as well as the code used to prep the data, can be used on any dataset, so feel free to experiment on this with your own datasets!

Each layer has been activated using a *sigmoid* activation function. A *momentum* parameter of 0.9 has been used, as is a standard rule of thumb. As outlined in the presentation, momentum is used to order to accelerate Stochastic Gradient Descent and thus train neural networks more efficiently. Each neural network has been trained for 2500 epochs, but this number can be modified according to your system specifications (increase it for faster systems/GPU's and vice versa).

Creating an AI

The code chunk below walks you through the various steps involved in building and training a simple double-hidden-layer neural network. For this sample we will assume 10 neurons in the first hidden layer and 5 in the second.

Suppose your training input data is called `train`. As discussed above, your input data should be scaled. The predictors are pulled into the matrix `x_train` and your response variable is pulled into the matrix `y_train`, as shown in the data prep code above. Then here's how it works:

```
model <- keras_model_sequential() #Step 1: Model is initialized as s
sequential object

model %>%

  #This adds the first hidden layer, which contains 10 neurons and uses a
sigmoid activation function. Note that    the input_shape argument for the
first hidden layer is always ncol(x_train), since this layer received its
input   from the input layer, which in turn comes from x_train
  layer_dense(units = 10, activation = "sigmoid", input_shape =
ncol(x_train)) %>%

  #This adds the second hidden layer, which has 5 neurons and uses a sigmoid
activation function. Here we do not    need to specify an input shape, since
it reads from the previous layer
  layer_dense(units = 5, activation = "sigmoid") %>%

  #This specifies the output layer, which has only 1 neuron since there is
only 1 response variable. If you have    more than one response this layer
```

```
will have that many neurons
  layer_dense(units = 1, activation = "sigmoid")

#Your neural network has been constructed. It is now time to pull it all
together.
model %>%
  compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.1,
momentum = 0.9), metrics = "mse")
#The above step is the most important, because this is where you define your
learning parameters. The loss function is the mean squared error. The
optimizer argument specifies your gradient descent algorithm. In this case, I
have used Stochastic Gradient Descent with a momentum accelerator, which is
highly recommended as discussed in the presentation. The learning rate has
been set to 0.1 for this example.

#Your neural network is now ready to be trained. Train it using the following
command.
fit <- fit(model, x = x_train, y = y_train, batch_size = 1000, epochs = 2500,
verbose = 1) #Setting verbose = 1 returns a plot that tracks the progress of
your neural network by tracing the value of the loss function through the
training progress.
```

Training ANN's : Finding optimal Network Architecture

Single-layer architectures: (33-40-1), (33-80-1), (33-100-1), (33-120-1) Double-layer architectures: (33-80-40-1), (33-100-60-1), (33-120-60-1) Initial parameters being tuned: batch size 8000, learning rate = 0.05

```
set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn1_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 40, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- (model %>% predict(x_test))
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
```

```
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn1_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn1_cv_mse <- mean(nn1_mse)
nn1_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn2_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 80, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn2_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn2_cv_mse <- mean(nn2_mse)
nn2_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn3_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
```

```
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 100, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn3_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn3_cv_mse <- mean(nn3_mse)
nn3_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn4_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn4_mse[i] <- sum((as.numeric(predictions_keras) -
```

```
test_target)^2)/nrow(test)
}

nn4_cv_mse <- mean(nn4_mse)
nn4_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn6_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 80, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 40, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn6_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn6_cv_mse <- mean(nn6_mse)
nn6_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn7_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
```

```
  model %>%
    layer_dense(units = 100, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn7_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn7_cv_mse <- mean(nn7_mse)
nn7_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn8_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  k_set_value(model$optimizer$lr, 0.05)
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
```

```
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn8_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn8_cv_mse <- mean(nn8_mse)
nn8_cv_mse
```

Note how easy it is to build and train neural networks using Keras! The only thing different in the above 8 code chunks is the number of neurons in the hidden layers. Everything else is the same. Keras has a very modular nature which makes it easy to learn and use.

Tuning Batch Size and Learning Rate

Batch Sizes tested: 3000, 8000, 10000 Learning Rates tested for each batch size: 0.01, 0.05, 0.1, 0.001

In total, 12 models being tested.

```
set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn9_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.01,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 3000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn9_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}
```

```r
nn9_cv_mse <- mean(nn9_mse)
nn9_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn10_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 3000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn10_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn10_cv_mse <- mean(nn10_mse)
nn10_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn11_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
```

```
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.1,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 3000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn11_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn11_cv_mse <- mean(nn11_mse)
nn11_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn12_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.01,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
```

```
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn12_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn12_cv_mse <- mean(nn12_mse)
nn12_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn13_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr =
0.001, momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 3000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn13_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn13_cv_mse <- mean(nn13_mse)
nn13_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn14_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
```

```
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn14_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn14_cv_mse <- mean(nn14_mse)
nn14_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn15_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.1,
momentum = 0.9), metrics = "mse")
```

```
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn15_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn15_cv_mse <- mean(nn15_mse)
nn15_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn16_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr =
0.001, momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 8000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn16_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn16_cv_mse <- mean(nn16_mse)
nn16_cv_mse
```

```r
set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn17_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.01,
momentum = 0.9), metrics = "mse")
  k_set_value(model$optimizer$lr, 0.01)
  fit <- fit(model, x = x_train, y = y_train, batch_size = 10000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn17_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn17_cv_mse <- mean(nn17_mse)
nn17_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn18_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
```

```
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.05,
momentum = 0.9), metrics = "mse")
  k_set_value(model$optimizer$lr, 0.05)
  fit <- fit(model, x = x_train, y = y_train, batch_size = 10000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn18_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn18_cv_mse <- mean(nn18_mse)
nn18_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn19_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.1,
momentum = 0.9), metrics = "mse")
  k_set_value(model$optimizer$lr, 0.1)
  fit <- fit(model, x = x_train, y = y_train, batch_size = 10000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
```

```
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn19_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn19_cv_mse <- mean(nn19_mse)
nn19_cv_mse

set.seed(5)
folds <- createFolds(train_data_nn$claimcst0, k = 5)
nn20_mse <- c()
for (i in 1:length(folds)) {
  train <- train_data_nn[-folds[[i]], ]
  test <- train_data_nn[folds[[i]], ]
  train <- as.data.frame(scale(train, center = min, scale = (max - min)))
  test <- as.data.frame(scale(test, center = min, scale = (max - min)))
  x_train <- as.matrix(train[predictors])
  y_train <- as.matrix(train[targetvar])
  x_test <- as.matrix(test[predictors])
  y_test <- as.matrix(test[targetvar])
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
    layer_dense(units = 60, activation = "sigmoid") %>%
    layer_dense(units = 1, activation = "sigmoid")
  model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr =
0.001, momentum = 0.9), metrics = "mse")
  fit <- fit(model, x = x_train, y = y_train, batch_size = 10000, epochs =
2500, verbose = 1)
  preds <- model %>% predict(x_test)
  predictions_keras <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  test_target <- (y_test * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])
  nn20_mse[i] <- sum((as.numeric(predictions_keras) -
test_target)^2)/nrow(test)
}

nn20_cv_mse <- mean(nn20_mse)
nn20_cv_mse
```

Test Data MSE

```
#Poisson-Gamma GLM
pois_glm <- glm(fre_glm_formula, data = train_data, family = poisson(link =
"log"), offset = log(exposure))
pois_preds <- predict(pois_glm, test_data, type = "response")

gam_glm <- glm(sev_glm_formula, data = train_data[train_data$severity > 0, ],
```

```
family = Gamma(link = "log"))
gam_preds <- predict(gam_glm, test_data, type = "response")

pois_gam_preds <- pois_preds * gam_preds
pois_gam_rmse <- sqrt(sum((pois_gam_preds -
test_data$claimcst0)^2)/nrow(test_data))

#Tweedie GLM
tweedie_glm <- glm(formula = tweedie_glm_formula, data = train_data, family =
tweedie(var.power = optimal_p, link.power = 0), offset = log(exposure))

tweedie_preds <- predict(tweedie_glm, test_data, type = "response")
tweedie_rmse <- sqrt(sum((tweedie_preds -
test_data$claimcst0)^2)/nrow(test_data))

#Neural Network
train_nn <- dummy.data.frame(train_data, names = factors)
train_nn <- as.data.frame(lapply(train_nn, as.numeric))
test_nn <- dummy.data.frame(test_data, names = factors)
test_nn <- as.data.frame(lapply(test_nn, as.numeric))

train_nn <- as.data.frame(scale(train_nn, center = min, scale = (max - min)))
test_nn <- as.data.frame(scale(test_nn, center = min, scale = (max - min)))

x_train <- as.matrix(train_nn[predictors])
y_train <- as.matrix(train_nn[targetvar])

x_test <- as.matrix(test_nn[predictors])
y_test <- as.matrix(test_nn[targetvar])

mlp <- keras_model_sequential()
mlp %>%
  layer_dense(units = 120, activation = "sigmoid", input_shape =
ncol(x_train)) %>%
  layer_dense(units = 60, activation = "sigmoid") %>%
  layer_dense(units = 1, activation = "sigmoid")
mlp %>%
  compile(loss = "mean_squared_error", optimizer = optimizer_sgd(lr = 0.1,
momentum = 0.9), metrics = "mse")
fit <- fit(mlp, x = x_train, y = y_train, batch_size = 3000, epochs = 2500,
verbose = 1)
preds <- (mlp %>% predict(x_test))
predictions_mlp <- (preds * (max(dataCar_active_nn[targetvar]) -
min(dataCar_active_nn[targetvar]))) + min(dataCar_active_nn[targetvar])

mlp_rmse <- sqrt(sum((predictions_mlp -
test_data$claimcst0)^2)/nrow(test_data))

test_rmse <- c(pois_gam_rmse, tweedie_rmse, mlp_rmse)
```

```r
models <- c("Poisson-Gamma", "Tweedie", "MLP")
test_rmse_df <- data.frame(models, test_rmse)
colnames(test_rmse_df) <- c("Model", "Test RMSE")
test_rmse_df
```

AIC

```r
n <- nrow(dataCar_active)
k_glm <- 6 #Number of rating factors in your GLM's
k_nn <- 33 #Exposure added as a predictor

aic <- function(n, model_predictions, k) {
  rss <- sum((model_predictions - test_data$claimcst0) ^ 2)
  aic <- (n * log(rss / n)) + (2 * k)
  return(aic)
}

#Poisson-Gamma Model
pg_aic <- aic(n, pois_gam_preds, (k_glm + 1))

#Tweedie GLM
tweedie_aic <- aic(n, tweedie_preds, (k_glm + 1))

#ANN
mlp_aic <- aic(n, predictions_mlp, (k_nn + 1))

#Compiling AIC Values
models <- c("Poisson-Gamma GLM", "Tweedie GLM", "ANN")
aics <- c(pg_aic, tweedie_aic, mlp_aic)
aic_results <- data.frame(models, aics)
colnames(aic_results) <- c("Model", "AIC")
aic_results
```
