Summary of ABC Analytic Project:

ABC Corporation has a simple business question, how can we identify patients that have a high risk of being readmitted to a hospital after being discharged? This report runs through the process of creating a model that will be able to identify patients that have a higher than average risk of being readmitted. The model makes use of Machine Learning and Neural Networks. In this report we go into detail on creating the analytic plan, the preliminary plan, and then a final plan. We also go into recommendations and a plan to roll this model into production. The summary of the Analytic Project will give a high-level overview of the process. The full details of the process are within this report as well, along with an addendum of the code used to generate the model.

The question ABC Corporation is asking themselves is binary classification problem. Will the patient be readmitted, YES or NO. With that question in mind, we know that the best Neural Network model to build is a binary model. Before building the model, we had to preprocess the data so that it would work in a Neural Network model. This consisted of cleaning the data and standardizing it. This process is important in machine learning because with out standardizing the data the model will not be able to train properly.

Once the pre-processing was complete, we were able to start building our first model. Going into this model we just wanted to train the data in a neural network and get a good foundation to build a final model. After working through the model structure and tweaking it a bit we were able to get a model that was training. While this models architecture ended up not working as well as we wanted, it did give us that starting

foundation that we were looking for. After running the model we knew that there were some changes that we needed to make in our final model.

The final model is an improvement on the first model, and the improvement was substantial. This model was much more accurate due to some key architecture changes. The model now had metrics that are trending in the correct direction. The accuracy was improving as the model learned and the loss was decreasing. After tinkering with this final model, we got it to be as efficient as possible. We now have a model that is able to give predictions on if a patient will be readmitted to the hospital or not and thus, giving us an answer to the business question.

Full Analysis:

The final recommendations sound be simple but require a lot of backend work to produce. Our recommendation is to roll out this model into production. We want to build an app that will be used by employees, this app will allow you to enter patient information and see how likely they were to be readmitted to the hospital. This way employees can plan for patients that have high readmittance rate. With this in place, we can reduce readmittance rates over time. Admittedly the model itself can use some more work on accuracy and loss statistics, but we would work all of that out before deploying the model to production.

The overall approach for this report was all based on a question posed by the ABC Corporation. They want to assess the risk of hospital readmittance when a patient is discharged. Our main objective is to create a model that looks at a patient admitted with one diabetes diagnosis and see if they have an above average chance of being

readmitted. The target is to find patients that have a higher-than-average rate of being readmitted to the hospital. The model will be a Neural Network and the problem itself is a classification problem. We built an analytic plan, preliminary model, and final model. In the analytic plan we explained what the process will be for the project. The preliminary model is testing our information and highlight what we can improve on from that preliminary model. Once we had a sufficient initial model, we will use that as our foundation when building a final model. The approach is following machine learning checks and balances, where models are created and tested for accuracy and loss. Our baseline is conservative, we are looking for accuracy above 50 % for accuracy and below 60% for loss. We aim to fine-tune and improve on these scores with time and practice.

The data set was directly from the ABC Corporation, it represents 10 years of clinical care throughout their 25 hospitals. The dataset consisted of 29 total columns and 68,358 entries. 2 of the columns are unique identifiers and they are excluded from the research. 15 of the columns are categorical data and 12 are numerical. The data is also only of patients that have a diabetes diagnosis. The feature engineering steps are explained in the following paragraph and stay consistent for both feed-forward Neural Networks built.

The preliminary results gave us a good baseline for what building and testing a neural network. To prepare the data for the preliminary results we had to do a couple of steps. The focus of the project is creating a model that can see what patients have a higher chance of hospital readmittance. To prepare the data for this problem we had to

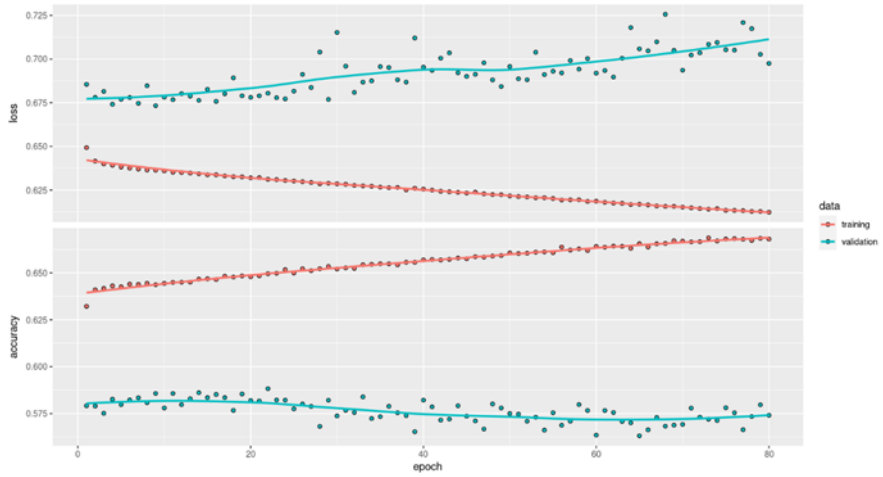
pre-process it. There was a column that was named readmitted, this was filled with data of “Yes” or “No”. This is what spawned the idea of building a binary type of model. The actual change to the data was turning all the yes observations to the number one and no observations to the number 0. This gives us the ability to create a binary classification model. With this done we can move on to one-hot encoding all the categorical data and standardize all of the numerical data. When implementing the one-hot encoding, we use the ‘dummyVars’ function in the ‘caret’ R library. The ‘dummyVars’ function take the categorical variables as inputs and outputs an encoder that knows how to encode the variables for any dataset. We also standardized our data in this pre-processing. Standardizing is an important step in most machine learning algorithms. “*In general, it isn’t safe to feed into a neural network data that takes relatively large values or data that is heterogeneous. Doing so can trigger large gradient updates that will prevent the network from converging.*” (Chollet, Francois. Deep Learning with R, Second Edition, Manning Publications, 2022, pp. 174–175.) When building Machine Learning models, we also need to split the data into training and test sets. The model is built with the training set data and tested with the test data. This is to avoid the model just memorizing the data and predicting everything perfectly. We need the data separate because when we do a test it should preform well on new data rather than just memorizing the old data. All of this was done before creating our first neural network.

The first neural network gave us insight of what we may need to fix and what we can keep in our model. The model was created with 4 hidden layers. The reason for 4 layers was because the data set had large dimensions and when you have a large data set more layers can be helpful. The first layer in the network had 420 units, the next had

150 units, layer three had 16 units, and the final layer had 1 unit. The units were chosen because when you give larger number of units it gives the model more freedom when learning the representations. Given that this was our preliminary model, this was the perfect opportunity to have large number of units. The first three layers had an activation function of “relu”. Relu is a popular activation function in deep learning, so it was a low risk choice for the preliminary report. The final layer had the activation function of “binary crossentropy”. Binary crossentropy is used for two-class classification problems. That is exactly the problem we are dealing with, making it an easy choice. The optimization algorithm was chosen for a similar reason to choosing relu as an activation function. “rmsprop” is a good default choice for most problems you can think of, so it was chosen as our optimization algorithm. All this effort was to get a model that overfit, then we would focus maximizing generalization.

The first thing we did to see how our model preformed was look at the plot that it generated. While this is not the best way to interpret how a model is preforming, it is still good to take a look.

Training and validation loss and accuracy metrics:

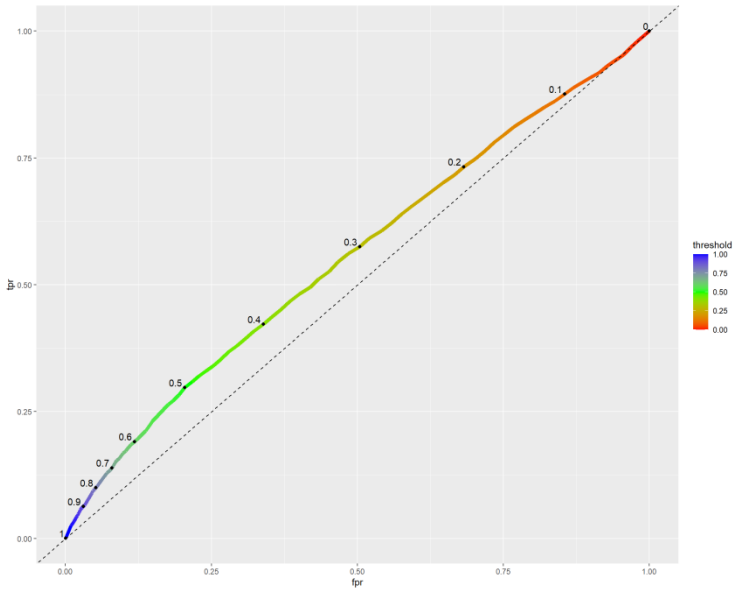


Model 1 code:

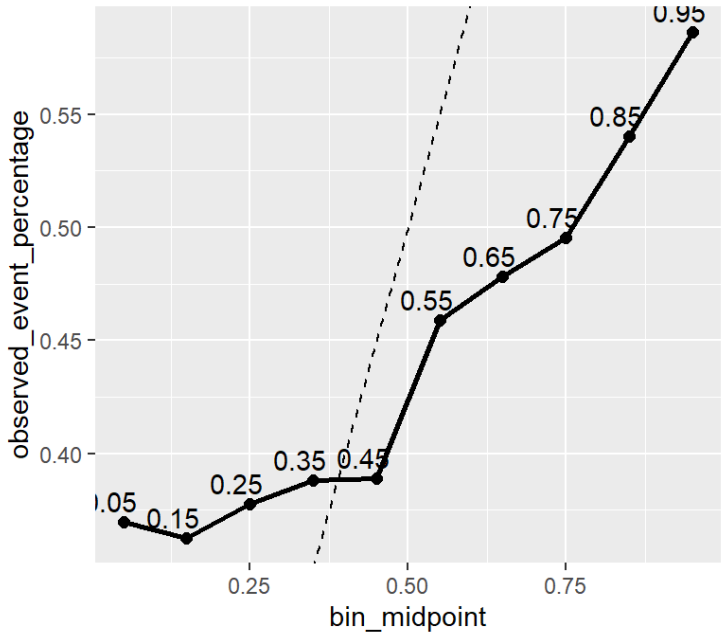
```
use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 420, activation = "relu"),
  layer_dense(units = 150, activation = "relu"),
  layer_dense(units = 16, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")
history <- fit(model, training_features, training_labels,
               epochs = 80, batch_size = 512, validation_split = 0.33)
plot(history)
```

Looking at our Training and validation loss and accuracy metrics, we see that the validation loss and validation accuracy are suffering. They are both trending in the wrong direction. This was the main thing that we wanted to fix in the final model, we must improve on the validation metrics. The final test of model 1 came with ROC curves, AUC, and calibration curves. In the preliminary results we did not run an AUC curve (area under ROC), but for the final we want to find show our AUC curve.

ROC Curve:



Calibration Curve:



Both the ROC curve and calibration curve need to show some improvement in the final report. The ROC curve is supposed to be as close to the top left corner as

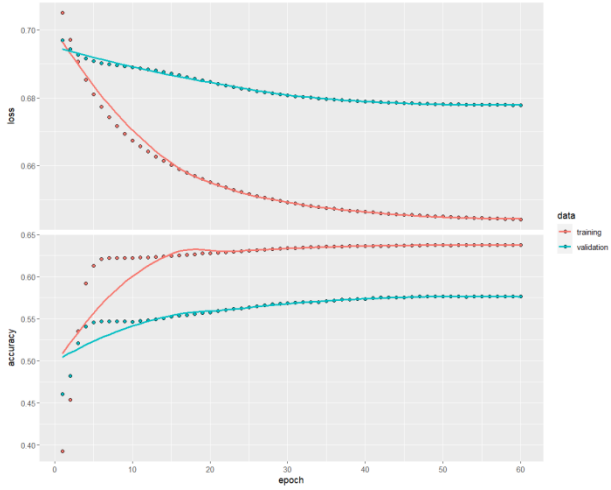
possible. The Calibration curve is supposed to be as close to the dotted line as possible. The main objectives in the final report are: 1,Try at least one additional dense feed-forward neural network with a different architecture. 2. Try to get a model that is capable of overfitting and then select the number of epochs using the learning curves. 3. Evaluate and compare the dense feed-forward neural networks using ROC curves, AUC, and calibration curves. 4. Discuss the findings from the above analyses, as well as how the model will be used in practice. 5. Discuss future research/steps forward. In the initial preliminary report, I mentioned early stop engineering, but that is outside the scope of this project and will not be performed. In the in Analytic plan, we also discussed running the data through logistic regression, LDA, KNN, and Trees to see how the neural network predictions compare to other Machine Learning methods. Due to time constraints, we will not be focusing on those other models, all time and effort will be spent on the Neural Network.

With all the information gathered from the preliminary report we can start our final report model. The feature engineering steps are the same as the preliminary report. The first thing we had to fix was the fact that our loss and accuracy curves were trending in the wrong direction. Loss was trending up and accuracy was trending down. The process to try and fix this began with increasing the layers and the units in the neural network. This was not successful, and the model did not improve. I then tried to change the epochs and batch size. None of the above was successful, but then I changed the optimizer to “adadelta” this was able to correct the trend of my model. “*Adadelta optimization is a stochastic gradient descent method that is based on adaptive learning rate per dimension to address two drawbacks. The continual decay of learning rates*

throughout learning and the need for a manually selected global learning rate. (Team, Keras. “Keras Documentation: Adadelta.” Keras, https://keras.io/api/optimizers/adadelta/.)

With this change to the model we were able to get the loss and accuracy rates to trend in the correct direction. That is, loss is decreasing, and accuracy is increasing. The training and validation metrics can be seen below. The improvement from model 1 is substantial, next we must see if it beats our baseline. While the model does not beat our arbitrary baseline set in the beginning of the report, it is drastically improving on the last model. After tinkering with the model for a bit the loss and accuracy was not changing no matter what I tried. The only change we can make is to have less epochs in the model. This is because it looks like the learning stalls around 45 epochs.

Training and validation loss and accuracy metrics:



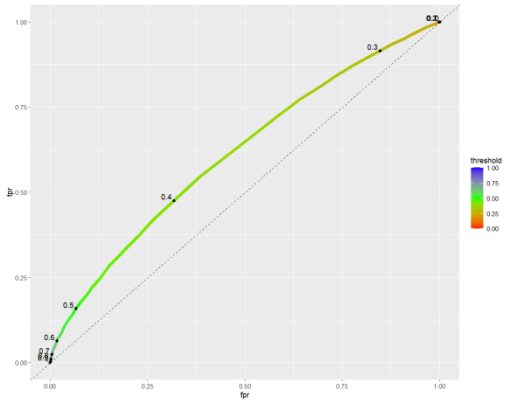
Code for Model 2.0:

Code for Model 1:

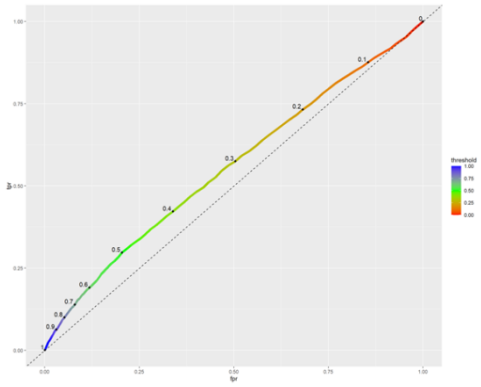
```
# Neural network # 2

use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 550, activation = "relu"),
  layer_dense(units = 250, activation = "relu"),
  layer_dense(units = 250, activation = "relu"),
  layer_dense(units = 100, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "adadelta",
        loss = "binary_crossentropy",
        metrics = "accuracy")
history <- fit(model, training_features, training_labels,
               epochs = 60, batch_size = 200, validation_split = 0.33)
plot(history)
```

```
use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 420, activation = "relu"),
  layer_dense(units = 150, activation = "relu"),
  layer_dense(units = 16, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")
history <- fit(model, training_features, training_labels,
               epochs = 80, batch_size = 512, validation_split = 0.33)
plot(history)
```

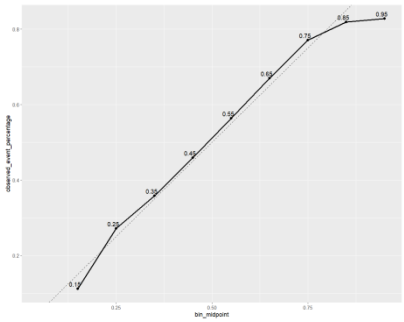
ROC Curve from Model 2.0:



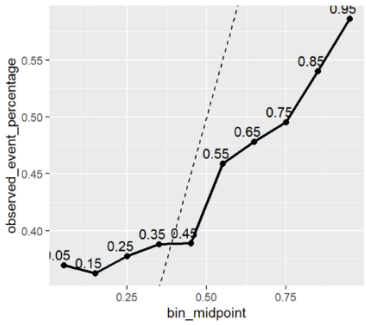
ROC Curve from Model 1:



Calibration Curve from Model 2.0:



Calibration Curve from Model 1:



We get a nice comparison of the 2 models evaluation metrics above. As you can see Model 2.0 improves on Model 1 in every way. The validation accuracy and loss are trending in the correct direction. The ROC curve is better leading to a better AUC curve, and the Calibration Curve from Model 2.0 is much better than the Calibration Curve from Model 1. Model 2.0 is better than Model 1, and we can now move on to the next steps of answering the business need.

The big question is how we can use this model to solve our business need. Again, our need is to create a model that can identify patients that have a higher than average risk of readmittance. By deploying this model, we will have way to predict what patients will have a higher chance of readmittance than average. The recommendation is to roll out this model into a product. What this means is we will be creating an app using Shiny that will put this model into use. This way we can have employees enter patient information to check on the chance of readmittance. Then employees can create a plan on how to deal with the situation. As mentioned, this data was only on patients that have diabetes. Moving forward I would like to retrain the model on all patient data and create subgroups of patients that have high readmittance rate. This way we can create a plan when we have a patient that fits the profile of high chance of readmittance. This way we will have a comprehensive plan for patients right when they walk in the hospital the first time. These recommendations should decrease our number of readmissions over time.

CODE (All of this is posted in Posit.Cloud as well):

```
# Install everything:
library(reticulate)
library(keras)
library(tensorflow)
virtualenv_create("my_tf_workspace")
use_virtualenv("my_tf_workspace", required = TRUE)
install_tensorflow(method="virtualenv",
                   envname="my_tf_workspace",
                   pip_options = "--no-cache-dir",
                   version = "cpu")
install_keras(method="virtualenv",
```

```
        envname="my_tf_workspace",
        pip_options = "--no-cache-dir",
        version = "cpu")
# Prelimanary report start
library(dplyr)
library(caret)
library(reticulate)
library(tensorflow)
library(keras)
library(MESS)

#Creating features
data <- project_data_1_
data$readmit_binary <- ifelse(data$readmitted == "yes",1,0)
# Taking out not needed data
data <- data[,-c(1,2)]

training_ind <- createDataPartition(data$readmit_binary,
                                    p = 0.75,
                                    list = FALSE,
                                    times = 1)
training_set <- data[training_ind, ]
test_set <- data[-training_ind, ]

## Categorical Data:
# CONVERT INTO FACTORS
data$race <- factor(data$race)
data$gender <- factor(data$gender)
data$age <- factor(data$age)
data$diag_1 <- factor(data$diag_1)
```

```
data$diag_2 <- factor(data$diag_2)

data$diag_3 <- factor(data$diag_3)

data$max_glu_serum <- factor(data$max_glu_serum)

data$A1Cresult <- factor(data$A1Cresult)

data$insulin <- factor(data$insulin)

data$change <- factor(data$change)

data$diabetesMed <- factor(data$diabetesMed)

data$readmitted <- factor(data$readmitted)

data$admission_source <- factor(data$admission_source)

data$admission_type <- factor(data$admission_type)

data$discharge_disposition <- factor(data$discharge_disposition)


#Set up onehot encoder

# Using all categorical columns below

#Confirming what columns are categorical:

names(Filter(is.factor,data))


# One hot coding the Training set

onehot_encoder <- dummyVars(~ race + gender + age + diag_1 + diag_2 + diag_3 + max_glu_serum +
A1Cresult + insulin + change + diabetesMed + readmitted + admission_source + admission_type +
discharge_disposition,

                            training_set[,
c("race","gender","age","diag_1","diag_2","diag_3","max_glu_serum","A1Cresult","insulin","change","d
iabetesMed","readmitted","admission_source","admission_type","discharge_disposition")],

                            levelsOnly = TRUE,

                            fullRank = TRUE)

onehot_enc_training <- predict(onehot_encoder,

                               training_set[,
c("race","gender","age","diag_1","diag_2","diag_3","max_glu_serum","A1Cresult","insulin","change","d
iabetesMed","readmitted","admission_source","admission_type","discharge_disposition")])

training_set <- cbind(training_set, onehot_enc_training)
```

```
# One hot coding the test set

onehot_enc_test <- predict(onehot_encoder, test_set[,
c("race","gender","age","diag_1","diag_2","diag_3","max_glu_serum","A1Cresult","insulin","change","d
iabetesMed","readmitted","admission_source","admission_type","discharge_disposition")])

test_set <- cbind(test_set, onehot_enc_test)




## Numerical Data

names(Filter(is.numeric,data))



# Here I used all of the numeric columns besides "readmit_binary. That was the binary column I created
to be able to run tests

#Standardize:

test_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")] <- scale(test_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")],


center = apply(training_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")], 2, mean),


scale = apply(training_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")], 2, sd))

training_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")] <- scale(training_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")])
```

```
training_features <- array(data = unlist(training_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")]),

                    dim = c(nrow(training_set), 33))# Not sure why I used 33 here, that was on the lab but it
was not explained.

training_labels <- array(data = unlist(training_set[, 28]),

                  dim = c(nrow(training_set)))

test_features <- array(data = unlist(test_set[,
c("time_in_hospital","num_lab_procedures","num_procedures","num_medications","number_outpatie
nt","number_emergency","num_medications_no","number_inpatient","number_diagnoses","num_med
ications_steady","num_medications_up","num_medications_down")]),

                  dim = c(nrow(test_set), 33))

test_labels <- array(data = unlist(test_set[, 28]),

                dim = c(nrow(test_set)))



# Build the neural network:


use_virtualenv("my_tf_workspace")


model <- keras_model_sequential(list(
  layer_dense(units = 420, activation = "relu"),
  layer_dense(units = 150, activation = "relu"),
  layer_dense(units = 16, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
      optimizer = "rmsprop",
      loss = "binary_crossentropy",
      metrics = "accuracy")
history <- fit(model, training_features, training_labels,
           epochs = 80, batch_size = 512, validation_split = 0.33)
```

```
plot(history)



# Build the ROC curve
predictions <- predict(model, test_features)
test_set$p_prob <- predictions[, 1]

over_threshold <- test_set[test_set$p_prob >= 0.35, ]
fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
fpr

tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
tpr

roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {
  over_threshold <- test_set[test_set$p_prob >= i, ]
  fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr
  tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr
}
ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))
```

```
# AUC Curve
auc <- auc(x = roc_data$fpr, y = roc_data$tpr, type = "spline")
auc

#.58

# Build the Calibration  curve

calibration_data <- data.frame(bin_midpoint=seq(0.05,0.95,0.1),
                               observed_event_percentage=0)
for (i in seq(0.05,0.95,0.1)) {
  in_interval <- test_set[test_set$p_prob >= (i-0.05) & test_set$p_prob <= (i+0.05), ]
  oep <- nrow(in_interval[in_interval$readmit_binary==1, ])/nrow(in_interval)
  calibration_data[calibration_data$bin_midpoint==i, "observed_event_percentage"] <- oep
}
ggplot(data = calibration_data, aes(x = bin_midpoint, y = observed_event_percentage)) +
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(size = 2) +
  geom_text(aes(label = bin_midpoint), hjust = 0.75, vjust = -0.5)



#_____________________________________________________=_____________________________
# Neural network # 2

use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 550, activation = "relu"),
  layer_dense(units = 250, activation = "relu"),
```

```
  layer_dense(units = 250, activation = "relu"),
  layer_dense(units = 100, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "adadelta",
        loss = "binary_crossentropy",
        metrics = "accuracy")
history <- fit(model, training_features, training_labels,
               epochs = 60, batch_size = 200, validation_split = 0.33)
plot(history)



results <- model %>% evaluate(test_features, test_labels)

results

# Neural network 2.1
use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 100, activation = "relu"),
  layer_dense(units = 75, activation = "relu"),
  layer_dense(units = 50, activation = "relu"),
  layer_dense(units = 25, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "adadelta",
        loss = "binary_crossentropy",
```

```
      metrics = "accuracy")
history <- fit(model, training_features, training_labels,
            epochs = 50, batch_size = 50, validation_split = 0.33)
plot(history)

# Build the ROC curve
predictions <- predict(model, test_features)
test_set$p_prob <- predictions[, 1]

over_threshold <- test_set[test_set$p_prob >= 0.35, ]
fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
fpr

tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
tpr

roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {
  over_threshold <- test_set[test_set$p_prob >= i, ]
  fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr
  tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr
}
ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
```

```
        aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# AUC Curve
auc <- auc(x = roc_data$fpr, y = roc_data$tpr, type = "spline")
auc
#.60

# Build the Calibration  curve
calibration_data <- data.frame(bin_midpoint=seq(0.05,0.95,0.1),
                               observed_event_percentage=0)
for (i in seq(0.05,0.95,0.1)) {
  in_interval <- test_set[test_set$p_prob >= (i-0.05) & test_set$p_prob <= (i+0.05), ]
  oep <- nrow(in_interval[in_interval$readmit_binary==1, ])/nrow(in_interval)
  calibration_data[calibration_data$bin_midpoint==i, "observed_event_percentage"] <- oep
}
ggplot(data = calibration_data, aes(x = bin_midpoint, y = observed_event_percentage)) +
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(size = 2) +
  geom_text(aes(label = bin_midpoint), hjust = 0.75, vjust = -0.5)



# Neural Network 3

use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 1500, activation = "relu"),
  layer_dense(units = 850, activation = "relu"),
```

```
  layer_dense(units = 250, activation = "relu"),
  layer_dense(units = 100, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model,
        optimizer = "sgd",
        loss = "binary_crossentropy",
        metrics = "accuracy")
history <- fit(model, training_features, training_labels,
               epochs = 30, batch_size = 200, validation_split = 0.33)
plot(history)

results <- model %>% evaluate(test_features, test_labels)
results

plot(history$metrics$val_loss, type = 'o',
     main = "Validation Loss for a Model with Appropriate Capacity",
     xlab = "Epochs", ylab = "Validation Loss")

# Build the ROC curve
predictions <- predict(model, test_features)
test_set$p_prob <- predictions[, 1]

over_threshold <- test_set[test_set$p_prob >= 0.35, ]
fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
fpr

tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
tpr
```

```
roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {
  over_threshold <- test_set[test_set$p_prob >= i, ]
  fpr <- sum(over_threshold$readmit_binary==0)/sum(test_set$readmit_binary==0)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr
  tpr <- sum(over_threshold$readmit_binary==1)/sum(test_set$readmit_binary==1)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr
}
ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# AUC Curve
auc <- auc(x = roc_data$fpr, y = roc_data$tpr, type = "spline")
auc

# Build the Calibration  curve
calibration_data <- data.frame(bin_midpoint=seq(0.05,0.95,0.1),
                               observed_event_percentage=0)
for (i in seq(0.05,0.95,0.1)) {
  in_interval <- test_set[test_set$p_prob >= (i-0.05) & test_set$p_prob <= (i+0.05), ]
  oep <- nrow(in_interval[in_interval$readmit_binary==1, ])/nrow(in_interval)
  calibration_data[calibration_data$bin_midpoint==i, "observed_event_percentage"] <- oep
}
ggplot(data = calibration_data, aes(x = bin_midpoint, y = observed_event_percentage)) +
```

```
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(size = 2) +
  geom_text(aes(label = bin_midpoint), hjust = 0.75, vjust = -0.5)

model %>% predict(test_features)
```