Ryan Canfield

Dr. Salemi

DSE6211

June 9th, 2024

Final Report

**Executive Summary**

This report will summarize and conclude our findings made after our last preliminary results meeting. After taking the comments into consideration that were given to us by our client in the last meeting, I have made another dense feedforward neural network with ways to prevent its overfitting. The methods that were used to prevent overfitting were early stoppage, and dropout. The results from this neural network are as follows; its accuracy was 86.06%, its sensitivity was 70.5%, and its specificity was 93.58%. This makes it the best model out of the three previous ones. The table below shows all the models with their updated values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Performance Metrics | Neural Network  (Preventing Overfitting) | Neural Network  (Overfitted) | Logistic Regression | Support Vector Machine |
| Accuracy | 86.06% | 85.58% | 80.16% | 86.05% |
| Sensitivity | 70.5% | 72.87% | 61.30% | 72.57% |
| Specificity | 93.58% | 91.71% | 89.26% | 92.55% |

Figure 1 Table of performance metrics of the four models.

The neural network’s confusion matrix indicated a higher proportion of true negatives than false negatives, suggesting good predictive ability in identifying canceled bookings. With this modified model our findings suggest that the neural network is the best option to accurately predict ABC hotel’s booking cancellations based on their given dataset of 35,000 hotel bookings. This is the best model from our experimentation and will outperform other supervised learning models like logistic regression, and support vector machines. I recommend deploying this neural network model as the primary tool for predicting booking cancellations due to its superior performance and high accuracy. Thanks to this model ABC Hotels will be able to proactively manage booking cancellations, and by targeting the high-risk bookings found by the model, ABC Hotels can mitigate cancellations, optimize revenue generation, and enhance the overall guest experience. Below is the approach I took and the data I used, a detailed version of our findings and our model evaluations, and finally further recommendations.

**Approach & Data**

The problem that ABC hotels gave us was identifying customers whose bookings were at high risk of cancellation. The hotel would then be able to proactively go after and target those bookings, with tailored advertisements and/or offers, to prevent them from being canceled. Due to the data and the nature of the problem I decided to use a supervised learning predictive model that would accurately estimate the probability of cancellation for each booking. The first step to creating the model was to review the data so I would know what the label would be and what the features were and what data types they had. After reviewing the data, I found that the data had over 35,000 rows and 17 columns. Each row contained information about a customer when they booked the room with what deals, and if they ended up canceling or not. From there I decided that the label or target for this for the supervised classification problem was the booking\_status column. The column contains two values either canceled or not canceled. These two values will be changed to numerical values. 0 to signify if a guest has canceled their reservation and a 1 will be used to signify if they have not. Next was the data preprocessing steps. The ones that I used to answer the business needs of ABC Hotels were as follows. First, I checked for missing values found in the dataset and if any were found I removed them. Since there was none upon checking, I did nothing to deal with them. Next, I normalized the dataset making sure that all the values in the dataset fell between 0 and 1. This helped our models and avoid overfitting due to any outlier which would contribute more than the majority of the rest of the data. Finally, I split the dataset into two sets, a training set, and a testing set. The training set had 80% of the data and was used to create our models, and the testing set contained 20% of the data and was used to evaluate how well the model performs. After doing some light statistical analysis, I found that most of the data was numerical, but the columns that were not had to be changed using one hot encoding. The reason why this style of encoding was used was to not impose a hierarchy on the data. Fifteen features were used when creating the models, all of them except for the “Booking\_ID” column since it was a unique identifier. After the original preprocessing, the next step completed was to address all the comments given in the analytic plan. One comment given was to change the date column to a season column. Instead of having a plethora of models. I took the advice and limited it down to four models, to really focus on finding the best neural network possible. The reason why I completed these steps was to simulate an estimation of its real-world performance. A variety of machine learning models were created to predict the booking status based on the given attributes. These models included logistic regression, support vector machine, a dense neural network model, and a neural network model that dealt with overfitting. The purpose of this was to find the best possible model while still giving us a wide array of options. I wanted to have a wide array of options to analyze because looking at them and choosing the best option would give ABC hotels the best chance to accurately predict booking cancellations. By prioritizing high risk bookings, they can use more of their resources to provide more tailored offers and advertisements in hopes of keeping people from calling them. By increasing this they will become more efficient and make more money since less people will cancel their books and get a refund.

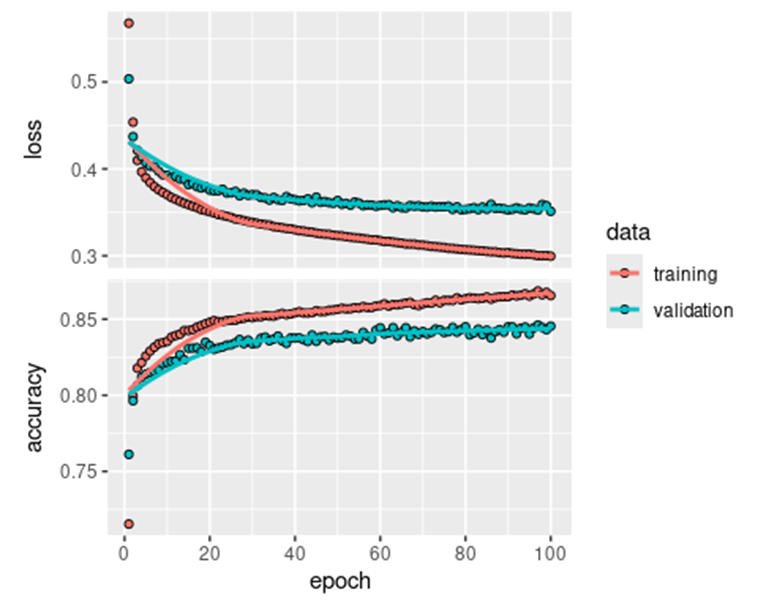
**Detailed Findings and Evaluation**

The results from the analysis showed that they would all yield positive results. The dense feedforward neural network with ways to prevent overfitting had an accuracy of 86.06%, its sensitivity was 70.5%, and its specificity was 93.58%. From the trained neural network model, it achieved an accuracy of 85.58% on the test set, with a sensitivity of 72.87% and specificity of 91.71%. The confusion matrix for all three models was able to provide insights into the model's performances. The neural network’s confusion matrix indicated a higher proportion of true negatives than false negatives, suggesting good predictive ability in identifying canceled bookings. Additionally, the other models yielded insightful results. Despite employing forward stepwise regression to identify the best possible features, the logistic regression appeared the least promising, likely due to having to fit many of its probabilities to zero or one as a result of one-hot encoding. The logistic regression model's metrics were an accuracy of 80.16% on the test set, with a sensitivity of 61.30% and a specificity of 89.26%. Conversely, the support vector machine demonstrated the highest performance, achieving an accuracy of 86.05% on the test set, with a sensitivity of 72.57% and a specificity of 92.55%. Below is a table comparing the performance of all the models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Performance Metrics | Neural Network  (Preventing Overfitting) | Neural Network  (Overfitted) | Logistic Regression | Support Vector Machine |
| Accuracy | 86.06% | 85.58% | 80.16% | 86.05% |
| Sensitivity | 70.5% | 72.87% | 61.30% | 72.57% |
| Specificity | 93.58% | 91.71% | 89.26% | 92.55% |

Figure 2 Table of performance metrics of the four models.

From this table we can see that the new neural network had the best performance out of all the models. I think it did the best due to all the changes we made. First, I performed PCA to limit the multicollinearity which was created by the one hot encoding. Next, I add stopping measures like early stopping and dropout to prevent the new neural network from overfitting on the data. With all these things added I believe that those were the reasons that it did better than the other NN and SVM models.



After building the model I plotted out the model’s accuracy and loss. This is important because it lets me better tune the model, and more easily communicate the performance of neural network models.

Figure 3 (right) loss and accuracy visual for the first NN model.

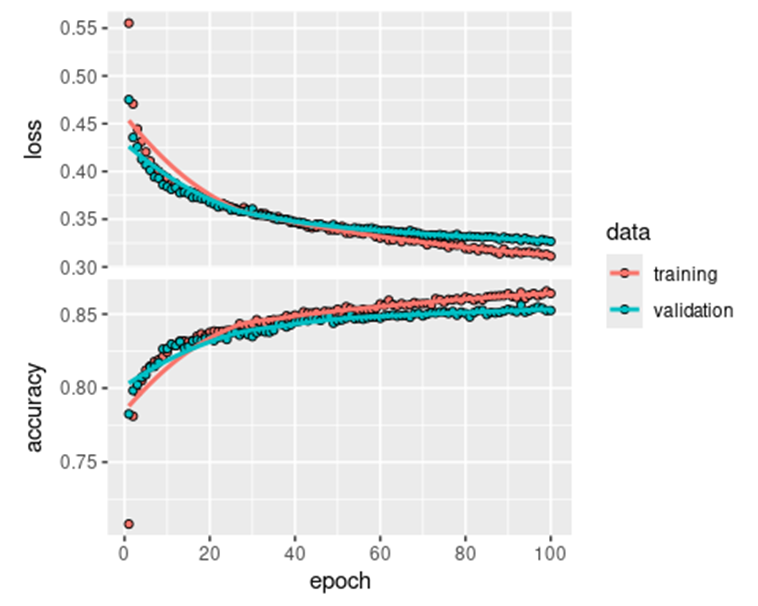
It also helps me make decisions on how to improve the model’s effectiveness and plan for future steps. As you can see from the graph above the validation line on the loss graph is above the training line the entire time for each epoch the validation loss starts at 0.44 and ends at 0.35, while the training line starts at 0.43 and ends at a 0.30. The opposite can be said for the accuracy graph. Here the validation line is below the training one, with it starting at 0.80 and ending at 0.84 while the training line starts at 0.84 and ends at 0.89. With the validation curve being higher on the loss and lower on the accuracy it shows signs of overfitting. To prevent this overfitting, I added coding chunks which took care of the overfitting which used early stoppage and drop out. Early stoppage is a technique used during the training to halt the training process before the model has fully converged which would prevent further overfitting after the training and validation line start to further divert from each other. Dropout is a regularization technique used in neural networks to reduce overfitting by randomly ignoring or dropping out a proportion of neurons during the training process. After completing this the new results are below.

Figure 4 (right) loss and accuracy visual for the second NN which accounts for overfitting.

As you can see from the visual to the right the training and validations are almost the same and they both appear to start and end at pretty much the same place. This shows that the model no longer overfits and is making predictions accurately on both the training and validation data sets. Below is a table comparing the two NN. From this table you can see the values of NN 2 are much closer than the values from NN 1, only being about 0.01 or 0.02 away from each other instead of 0.05.

|  |  |  |  |
| --- | --- | --- | --- |
| NN 1 Overfitting |  | Loss | Accuracy |
|  | Validation | 0.44 - 0.35 | 80% - 84% |
|  | Training | 0.43 - 0.30 | 84% - 89% |
| NN 2 no Overfitting |  |  |  |
|  | Validation | 0.45 - 0.33 | 0.80 - 0.85 |
|  | Training | 0.45 - 0.31 | 0.79 - 0.86 |

Figure 5 (left) table of values for loss and accuracy graph.

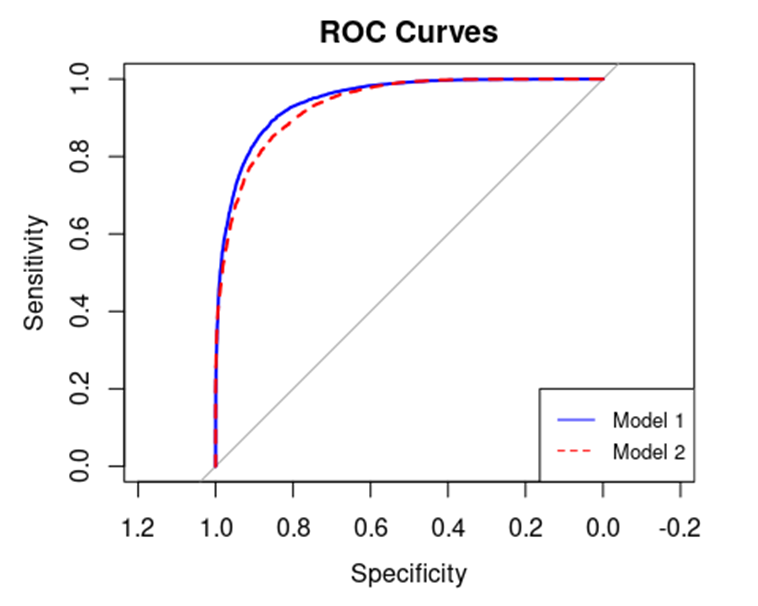


Figure 6 ROC curve graph between NN1 and NN2 showing their sensitivity and specificity rates.

This last visual shows the two ROC curves from both models. Here we can see that Model 2 has performed better, which is also backed up by figure 5 which is the table above.

**Recommendations**

This report concludes our analysis and findings following our recent review meeting. In response to client feedback, I revised the dense feedforward neural network, incorporating techniques to prevent overfitting such as early stopping and dropout. The results demonstrate significant improvements in model performance, with the neural network achieving an accuracy of 86.06%, sensitivity of 70.5%, and specificity of 93.58%, surpassing all other models evaluated. The neural network was optimized to predict booking cancellations for ABC Hotels and emerged as the preferred model due to its superior accuracy and robust predictive ability. By using this model, ABC Hotels can manage booking cancellations by identifying high-risk bookings and implementing targeted strategies, such as personalized offers and enhanced customer engagement. This approach not only mitigates cancellations but also optimizes revenue generation and enhances guest satisfaction. The final model successfully addresses the business question posed by ABC Hotel by using supervised learning techniques. I was able to accurately estimate the probability of cancellation for each booking, enabling ABC Hotels to shift around resources to strategically prioritize high-risk bookings. The model's performance metrics, accuracy, sensitivity, and specificity validate its suitability as a predictive tool tailored to the hotel's operational needs. Looking ahead, continual monitoring and retraining of the neural network model with updated data will ensure its effectiveness over time. Additionally, maybe looking into ensemble methods or further fine-tuning hyperparameters could potentially enhance predictive performance even further. It is also advisable to integrate real-time data feeds to enhance the model's responsiveness to changing booking dynamics and market conditions on the fly. In conclusion, the neural network provided will be a great asset for ABC Hotels. This will let them be able to make well informed decisions, optimize operations, and enhance guest experiences by proactively managing cancellation.

***Code Appendix***

DSE6211 Preliminary Results

Ryan Canfield

2024-06-11

## R Markdown

library(glmnet) #For a Logistic Regression model.

## Loading required package: Matrix

## Loaded glmnet 4.1-8

# Addressed comment and removed extra models  
library(MASS)  
library(e1071) # For a Support Vector Machine model.  
library(neuralnet) # For a basic Neural Network model.  
library(keras) # For a TensorFlow Neural Network model.  
library(tensorflow) # For a TensorFlow framework.  
library(reticulate) # For a python interface.  
library(caret) # For model training and evaluation.

library(ROCR) # For ROC analysis.

library(tidyverse) # For data manipulation.

library(ggplot2) # For data visualizations.   
library(RColorBrewer) # For coloring visuals.  
library(forcats) # For Manipulating and working with categorical variables.   
library(pROC) # For Roc curves.

library(yardstick) # For Calibration curves.

# Reading the CSV file into a data frame  
df <- read.csv("project\_data.csv")  
  
# Display a preview of the data frame.  
head(df)

## Booking\_ID no\_of\_adults no\_of\_children no\_of\_weekend\_nights no\_of\_week\_nights  
## 1 INN00001 2 0 1 2  
## 2 INN00002 2 0 2 3  
## 3 INN00003 1 0 2 1  
## 4 INN00004 2 0 0 2  
## 5 INN00005 2 0 1 1  
## 6 INN00006 2 0 0 2  
## type\_of\_meal\_plan required\_car\_parking\_space room\_type\_reserved lead\_time  
## 1 meal\_plan\_1 0 room\_type1 224  
## 2 not\_selected 0 room\_type1 5  
## 3 meal\_plan\_1 0 room\_type1 1  
## 4 meal\_plan\_1 0 room\_type1 211  
## 5 not\_selected 0 room\_type1 48  
## 6 meal\_plan\_2 0 room\_type1 346  
## arrival\_date market\_segment\_type repeated\_guest no\_of\_previous\_cancellations  
## 1 10/2/2017 offline 0 0  
## 2 11/6/2018 online 0 0  
## 3 2/28/2018 online 0 0  
## 4 5/20/2018 online 0 0  
## 5 4/11/2018 online 0 0  
## 6 9/13/2018 online 0 0  
## no\_of\_previous\_bookings\_not\_canceled avg\_price\_per\_room  
## 1 0 65.00  
## 2 0 106.68  
## 3 0 60.00  
## 4 0 100.00  
## 5 0 94.50  
## 6 0 115.00  
## no\_of\_special\_requests booking\_status  
## 1 0 not\_canceled  
## 2 1 not\_canceled  
## 3 0 canceled  
## 4 0 canceled  
## 5 0 canceled  
## 6 1 canceled

# Looking at different histograms to get a better idea of the data  
par(mfrow = c(2,3))  
hist(df$booking\_status, xlab = "Booking Status", main = "Booking Status Distribution")  
hist(df$avg\_price\_per\_room, col = 4, breaks = 20, xlab = "Room Price", main = "Average Price Distribution")  
hist(df$room\_type\_reserved, col = 3, xlab = "Room Type", main = "Room Type Distribution")  
hist(df$type\_of\_meal\_plan, col = 2, breaks = 10, xlab = "Meal Plan #", main = "Meal Plan Distrbution")  
hist(df$no\_of\_weekend\_nights, col = 5, xlab = "Weekend Nights", main = "Weekend Nights Distrbution")  
hist(df$no\_of\_week\_nights , col = 6, breaks = 8, xlab = "Week Nights", main = "Week Nights Distrbution")

A group of graphs showing different types of data

Description automatically generated

# Looking at different histograms to get a better idea of the data  
par(mfrow = c(2,3))  
hist(df$no\_of\_adults, col = 12, breaks = 4, xlab = "# of Adults", main = "Adult Distribution")  
hist(df$no\_of\_children, col = 7, breaks = 15, xlab = "# of Children ", main = "Children Distribution")  
hist(df$required\_car\_parking\_space, col = 8, breaks = 4, xlab = "No or Yes", main = "Do They Need a Parking Space?")  
hist(df$lead\_time, col = 9, breaks = 10, xlab = "Lead Time", main = "Distrbution of Lead Time")  
hist(df$repeated\_guest, col = 10, breaks = 4, xlab = "No or Yes", main = "Are They a Repeated Guest?")  
hist(df$no\_of\_previous\_cancellations, col = 11, breaks = 4, xlab = "No or Yes", main = "Have They Canceled Before?")

# For the proposed supervised classification, we can define the level "cancelled" as the positive class and "not cancelled" as the negative class. Then, once we build a classification model (for example, using a neural network) to predict booking\_status, we can use the predicted probability of positive class (i.e., "cancelled") membership as the risk cancellation score. This predicted probability is between 0 and 1. Defining these classes as the positive and negative classes, respectively, also provides the appropriate context for evaluating the predicted probabilities using ROC curves and AUC (which we will cover during an upcoming week).

### Preprocessing and addressing comments from Analytic Plan.  
  
# Checking for missing values.  
print(sum(is.na(df)))

## [1] 0

# Addresses comment.  
# Excluding Booking\_ID since it is a unique identifier   
# Assuming 'df' is your dataframe  
df <- df[, !(names(df) %in% "Booking\_ID")]  
  
  
  
  
# Addresses comment.  
# Changes arrival date to season so when we use One Hot Encoding.   
# It doesn't create hundreds of extra columns, dimensionality, and space.  
df$arrival\_date <- as.Date(df[, "arrival\_date"], format = "%m/%d/%Y")   
df$season <- ifelse(month(df$arrival\_date) %in% c(12, 1, 2), "Winter",  
 ifelse(month(df$arrival\_date) %in% c(3, 4, 5), "Spring",  
 ifelse(month(df$arrival\_date) %in% c(6, 7, 8), "Summer",  
 ifelse(month(df$arrival\_date) %in% c(9, 10, 11), "Fall", NA))))  
df$arrival\_date <- df$season  
df <- df[, -which(names(df) == "season")]  
  
# This changes our target variable to numerical values without emposing heiracrhy or an extra column.  
df$booking\_status <- ifelse(df$booking\_status == "not\_canceled", 1, 0)

# Changed from label encoding to One Hot encoding  
for (col in names(df)) {   
 if (!is.numeric(df[[col]])) {  
 # Perform one-hot encoding  
 df <- cbind(df, model.matrix(~df[[col]] - 1))  
 df <- df[, -which(names(df) == col)]  
 cat("One-hot encoding applied to column:", col, "\n")  
 }  
}

## One-hot encoding applied to column: type\_of\_meal\_plan   
## One-hot encoding applied to column: room\_type\_reserved   
## One-hot encoding applied to column: arrival\_date   
## One-hot encoding applied to column: market\_segment\_type

# Split the dataset into a training and testing set with an 80 - 20 split  
set.seed(123)  
ind <- createDataPartition(df$booking\_status, p = 0.8, list = FALSE)  
train <- df[ind, ]   
test <- df[-ind, ]  
  
# Scaling the data  
# Define indices of columns to be scaled  
columns\_to\_scale <- setdiff(seq\_len(ncol(train)), 12)  
  
# Scale selected columns in the training set  
train[, columns\_to\_scale] <- lapply(train[, columns\_to\_scale], scale)  
  
# Scale selected columns in the test set  
test[, columns\_to\_scale] <- lapply(test[, columns\_to\_scale], scale)  
  
remove\_prefix <- function(df) {  
 colnames(df) <- gsub("df\\[\\[col\\]\\]", "", colnames(df))  
 return(df)  
}  
  
train <- remove\_prefix(train)  
test <- remove\_prefix(test)  
  
# Convert training and test features to arrays  
training\_features <- array(data = unlist(train[, -12]),  
 dim = c(nrow(train), 31))  
test\_features <- array(data = unlist(test[, -12]),  
 dim = c(nrow(test), 31))  
  
# Convert training and test labels to arrays  
training\_labels <- array(data = unlist(train[, 12]),  
 dim = c(nrow(train)))  
test\_labels <- array(data = unlist(test[, 12]),  
 dim = c(nrow(test)))

library(reticulate)  
library(tensorflow)  
library(keras)  
  
use\_virtualenv("my\_tf\_workspace")

model <- keras\_model\_sequential(list(  
 layer\_dense(units = 100, activation = "relu"),  
 layer\_dense(units = 100, activation = "relu"),  
 layer\_dense(units = 10, activation = "relu"),  
 layer\_dense(units = 1, activation = "sigmoid")  
))

compile(model,  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = "accuracy")

NN <- fit(model, training\_features, training\_labels,  
 epochs = 100, batch\_size = 512, validation\_split = 0.33)  
  
plot(NN)

A graph of data and data

Description automatically generated

predictions4 <- predict(model, test\_features)  
head(predictions4, 10)

## [,1]  
## [1,] 0.755750775  
## [2,] 0.990785718  
## [3,] 0.984088659  
## [4,] 0.733701348  
## [5,] 0.949197471  
## [6,] 0.998743713  
## [7,] 0.995375335  
## [8,] 0.996213138  
## [9,] 0.998976588  
## [10,] 0.008950665

predicted\_class4 <- (predictions4[, 1] >= 0.5) \* 1  
head(predicted\_class4, 10)

## [1] 1 1 1 1 1 1 1 1 1 0

conf\_matrix4 <- confusionMatrix(as.factor(predicted\_class4), as.factor(test\_labels))  
conf\_matrix4

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 1744 387  
## 1 615 4501  
##   
## Accuracy : 0.8617   
## 95% CI : (0.8536, 0.8696)  
## No Information Rate : 0.6745   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6771   
##   
## Mcnemar's Test P-Value : 7.434e-13   
##   
## Sensitivity : 0.7393   
## Specificity : 0.9208   
## Pos Pred Value : 0.8184   
## Neg Pred Value : 0.8798   
## Prevalence : 0.3255   
## Detection Rate : 0.2407   
## Detection Prevalence : 0.2941   
## Balanced Accuracy : 0.8301   
##   
## 'Positive' Class : 0   
##

# Perform PCA on the training features  
pca\_model <- prcomp(training\_features, center = TRUE, scale. = TRUE)  
  
# Transform the training features  
training\_features\_pca <- predict(pca\_model, training\_features)  
test\_features\_pca <- predict(pca\_model, test\_features)  
  
# keeping components that explain 95% of the variance  
explained\_variance <- summary(pca\_model)$importance[2,]  
num\_components <- which(cumsum(explained\_variance) >= 0.95)[1]  
  
# Subset the transformed features to keep only the required number of components  
training\_features\_pca <- training\_features\_pca[, 1:num\_components]  
test\_features\_pca <- test\_features\_pca[, 1:num\_components]  
  
# Convert the matrices to arrays  
training\_features\_array <- array(training\_features\_pca, dim = c(nrow(training\_features\_pca), num\_components))  
test\_features\_array <- array(test\_features\_pca, dim = c(nrow(test\_features\_pca), num\_components))  
  
# Print the summary of the PCA model to understand the variance explained by each principal component  
summary(pca\_model)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 1.8276 1.59711 1.47457 1.3328 1.26138 1.22313 1.17039  
## Proportion of Variance 0.1077 0.08228 0.07014 0.0573 0.05133 0.04826 0.04419  
## Cumulative Proportion 0.1077 0.19003 0.26017 0.3175 0.36880 0.41706 0.46125  
## PC8 PC9 PC10 PC11 PC12 PC13 PC14  
## Standard deviation 1.14166 1.11840 1.09342 1.0491 1.02652 1.01342 1.00122  
## Proportion of Variance 0.04205 0.04035 0.03857 0.0355 0.03399 0.03313 0.03234  
## Cumulative Proportion 0.50329 0.54364 0.58221 0.6177 0.65171 0.68484 0.71717  
## PC15 PC16 PC17 PC18 PC19 PC20 PC21  
## Standard deviation 0.98368 0.97969 0.9350 0.92462 0.90698 0.90321 0.8315  
## Proportion of Variance 0.03121 0.03096 0.0282 0.02758 0.02654 0.02632 0.0223  
## Cumulative Proportion 0.74839 0.77935 0.8075 0.83512 0.86166 0.88798 0.9103  
## PC22 PC23 PC24 PC25 PC26 PC27  
## Standard deviation 0.81513 0.76412 0.68515 0.64896 0.57528 0.55816  
## Proportion of Variance 0.02143 0.01883 0.01514 0.01359 0.01068 0.01005  
## Cumulative Proportion 0.93171 0.95055 0.96569 0.97927 0.98995 1.00000  
## PC28 PC29 PC30 PC31  
## Standard deviation 2.367e-14 8.893e-15 5.869e-15 3.441e-15  
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00  
## Cumulative Proportion 1.000e+00 1.000e+00 1.000e+00 1.000e+00

# Print the number of components chosen  
print(num\_components)

## PC23   
## 23

model2 <- keras\_model\_sequential(list(  
 layer\_dense(units = 100, activation = 'relu'),  
 layer\_dropout(rate = 0.2),  
 layer\_dense(units = 100, activation = 'relu'),  
 layer\_dropout(rate = 0.2),  
 layer\_dense(units = 10, activation = 'relu'),  
 layer\_dropout(rate = 0.2),  
 layer\_dense(units = 1, activation = 'sigmoid')))  
  
compile(model2,  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = "accuracy")  
  
# Define early stopping callback  
early\_stopping <- callback\_early\_stopping(  
 monitor = "val\_loss",   
 patience = 10,   
 restore\_best\_weights = TRUE   
)  
  
  
  
NN\_pca <- fit(model2, training\_features\_pca, training\_labels,  
 epochs = 100, batch\_size = 512, validation\_split = 0.33, callbacks = list(early\_stopping)  
)  
  
plot(NN\_pca)

A graph of data and data

Description automatically generated

predictions5 <- predict(model2, test\_features\_pca)  
head(predictions5, 10)

## [,1]  
## [1,] 0.67554539  
## [2,] 0.98651737  
## [3,] 0.94410813  
## [4,] 0.67643857  
## [5,] 0.84987718  
## [6,] 0.90462512  
## [7,] 0.99052203  
## [8,] 0.97056472  
## [9,] 0.99886382  
## [10,] 0.05442176

predicted\_class5 <- (predictions5[, 1] >= 0.5) \* 1  
head(predicted\_class5, 10)

## [1] 1 1 1 1 1 1 1 1 1 0

conf\_matrix5 <- confusionMatrix(as.factor(predicted\_class5), as.factor(test\_labels))  
conf\_matrix5

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 1663 314  
## 1 696 4574  
##   
## Accuracy : 0.8606   
## 95% CI : (0.8524, 0.8685)  
## No Information Rate : 0.6745   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6687   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.7050   
## Specificity : 0.9358   
## Pos Pred Value : 0.8412   
## Neg Pred Value : 0.8679   
## Prevalence : 0.3255   
## Detection Rate : 0.2295   
## Detection Prevalence : 0.2728   
## Balanced Accuracy : 0.8204   
##   
## 'Positive' Class : 0   
##

# Predict probabilities for both models on test data  
pred\_model <- model %>% predict(training\_features)  
pred\_model2 <- model2 %>% predict(training\_features\_pca)  
  
pred\_model <- as.vector(pred\_model)  
pred\_model2 <- as.vector(pred\_model2)  
  
  
# Calculate ROC curves and AUC  
roc\_model <- roc(training\_labels, pred\_model)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

roc\_model2 <- roc(training\_labels, pred\_model2)

## Setting levels: control = 0, case = 1  
## Setting direction: controls < cases

# Plot ROC curves  
plot(roc\_model, col = "blue", lwd = 2, main = "ROC Curves")  
plot(roc\_model2, col = "red", lwd = 2, add = TRUE, lty = 2)  
legend("bottomright", legend = c("Model 1", "Model 2"), col = c("blue", "red"), lty = 1:2, cex = 0.8)

A graph of a model

Description automatically generated