# D213 - Advanced Data Analytics

Sentiment Analysis using Neural Networks

Sean P. Murphy

05 July 2022

## Contents

## Introduction

Prior to beginning the assigned sentiment analysis, we will provide some sense of the how this assignment fits into the overall context of our coursework so far and also prepare the working environment for project execution.

### Assignment Background

In our previous coursework, we were tasked by a popular hospital chain with analyzing several datasets, each with a slightly different focus. Initially, we identified criteria to quantify risk levels for patient readmission within one month of their initial discharge. When provided with patient prescription histories, we uncovered patterns that revealed the potential for medication interaction risks. Using patient survey data, we uncovered the existence of two specific groups of patients with distinct and diverging priorities so that the hospital can tailor individual marketing and patient care strategies more appropriately. Lastly, we provided daily revenue benchmarks against which policymakers can compare hospital revenue after implementing changes in policy or strategy to assess their relative success.

For this assignment, we will not be working with hospital-supplied data even though the project itself (as presented via the assignment introduction video) is still being commissioned by hospital stakeholders. This presentation is intended to be a proof-of-concept to demonstrate that a deep neural net approach to sentiment analysis using natural language processing models is a viable option for producing accurate and actionable analysis as part of future data analytic campaigns.

### Initial Preparation

```r
# Visualization Formatting
library(gridExtra)
library(viridisLite)

# Visualization Font Import
library(sysfonts)
library(showtextdb)
library(showtext)
font_add("LM Roman 10", "./lmroman10-regular.otf")
```

We will be loading a number of required libraries to perform some simple data manipulation (*dplyr* and *purr*), create visualizations (*ggplot2*), and to perform the bulk of the sentiment analysis (*keras*, *mlbench*, *neuralnet*).

```r
# Data Manipulation and Visualization
library(tidyverse)
library(magrittr)

# ML Libraries
library(keras)
library(mlbench)
library(tidytext)
library(neuralnet)
```

# Part I: Research Scope

## A1: Research Question

> Can a neural network applied to natural language processing extract a writer's sentiment with sufficient accuracy to allow the hospital to add a free response answer section to their standard surveys?

Unlike previous assignments, we are not addressing a research question directly connected to hospital business interests. We have previously been given patient demographics, admissions data, survey data, medication histories, or revenue statistics all taken directly from hospital databases. However, for this project, we are only asked to demonstrate a proof of concept in order to establish the potential usefulness of sentiment analysis empowered by natural language processing via neural networks. If successful, we intend to recommend that the hospital allow a free response section in future patient surveys.

## A2: Research Objective

> Create, train, and test a deep neural network model using natural language processing methods in order to assess writer sentiment (either positive or negative) given the text of a customer review.

Because we are seeking to answer the question of whether a deep neural network can be effective for empowering natural language processing in order to conduct sentiment analysis, we will frame our research objective accordingly. Our stakeholders have asked us to demonstrate that this approach to analysis will effectively address their future needs, so our goal is only to show a simple effective model that classifies user sentiment with an actionable level of accuracy.

## A3: Research Mechanism

> Sequential Artificial Neural Network

We will utilize a sequential artificial neural network for this project. When paired with an embedding layer in Keras, this model is ideal for natural language processing applications and supports custom word embedding (Saxena, 2021).

## Part II: Data Preparation

We will first import our selected data sets. This labeled data is similarly formatted and comes from three different sources: Amazon product reviews, IMDb movie reviews, and Yelp business reviews. Each file contains the original text of the respective reviews and their human-assessed sentiment: positive or negative.

### Data Import

We will be using the *readr* function imported as part of the *tidyverse*. There are no headers, so we will ensure that each row is considered as an observation and add our own labels at a later step. Each dataset will be imported individually and combined into a single dataframe afterward.

```
df.imdb   <- read_csv('./imdb_labelled.csv', col_names = FALSE)
df.yelp   <- read_csv('./yelp_labelled.csv', col_names = FALSE)
df.amazon <- read_csv('./amazon_cells_labelled.csv', col_names = FALSE)
```

### Data Concatenation

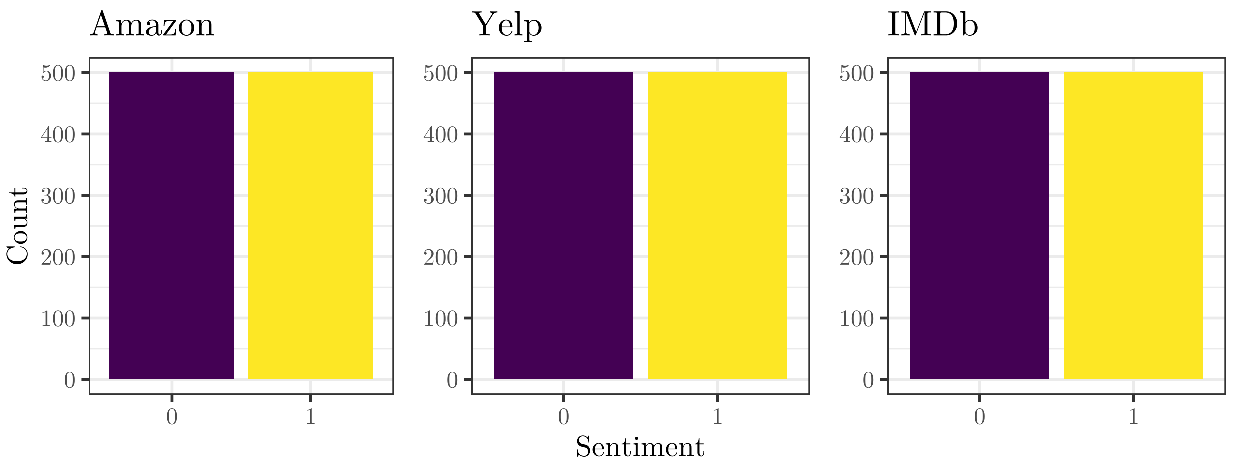
We will now combine each of these data frames together using *rbind* and add appropriate column labels. We will also print a *summary* of the resulting dataframe to examine its properties.

```
df <- rbind(df.amazon, df.yelp, df.imdb)
colnames(df) <- c('Review', 'Sentiment')
df$Sentiment <- as.factor(df$Sentiment)
df$Index <- c(1:3000)
summary(df[,c(1:2)])
```

```
##     Review           Sentiment
##  Length:3000        0:1500
##  Class :character   1:1500
##  Mode  :character
```

### B1: Exploratory Data Analysis

There are an equal number of positive and negative observations across the product, movie, and business reviews. This will help ensure that we do not produce skewed results from a selection bias.



Now we will give our data a cursory examination prior to designing the neural network that we will use to classify the sentiment of each review using natural language processing.

**Amazon Data:** For context and our general awareness, we will examine the first five rows of the Amazon data set.

```
head(df[1:5,])
```

```
## # A tibble: 5 x 3
##   Review                                                   Sentiment Index
##   <chr>                                                    <fct>     <int>
## 1 So there is no way for me to plug it in here in the US unless~ 0      1
## 2 Good case, Excellent value.                              1             2
## 3 Great for the jawbone.                                   1             3
## 4 Tied to charger for conversations lasting more than 45 minute~ 0      4
## 5 The mic is great.                                        1             5
```

**Yelp Data:** Similarly, we will examine the first five rows of observations from the Yelp dataset.

```
head(df[1001:1005,])
```

```
## # A tibble: 5 x 3
##   Review                                                   Sentiment Index
##   <chr>                                                    <fct>     <int>
## 1 Wow... Loved this place.                                 1          1001
## 2 Crust is not good.                                       0          1002
## 3 Not tasty and the texture was just nasty.                0          1003
## 4 Stopped by during the late May bank holiday off Rick Steve re~ 1   1004
## 5 The selection on the menu was great and so were the prices.    1   1005
```

**IMDb Data:** Finally, we will examine the first five rows of movie review data from the IMDb dataset.

```
head(df[2001:2005,])
```

```
## # A tibble: 5 x 3
##   Review                                                   Sentiment Index
##   <chr>                                                    <fct>     <int>
## 1 A very, very, very slow-moving, aimless movie about a distres~ 0   2001
## 2 Not sure who was more lost - the flat characters or the audie~ 0   2002
## 3 Attempting artiness with black & white and clever camera angl~ 0   2003
## 4 Very little music or anything to speak of.               0          2004
## 5 The best scene in the movie was when Gerardo is trying to fin~ 1   2005
```

### Lowercase Characters

In order to standardize the characters and ensure that like words are counted together, we will force all letters to lowercase characters. This will also decrease the vector space required to store our data which will reduce the overall computational overhead.

**Before:** As the data sets were provided, there is no standard format for user-generated input. Most people use sentence case standards, this is neither consistent nor useful for our purposes.

```
head(df, n=3)
```

```
## # A tibble: 3 x 3
##   Review                                                   Sentiment Index
##   <chr>                                                    <fct>     <int>
## 1 So there is no way for me to plug it in here in the US unless~ 0             1
## 2 Good case, Excellent value.                              1             2
## 3 Great for the jawbone.                                   1             3
```

**Conversion:** We will use the *tolower* function as part of the base R language to convert all character data to lowercase letters.

```
df$Review <- tolower(df$Review)
```

**After:** Now, we see below that all letters are represented using lower case characters.

```
head(df, n=3)
```

```
## # A tibble: 3 x 3
##   Review                                                   Sentiment Index
##   <chr>                                                    <fct>     <int>
## 1 so there is no way for me to plug it in here in the us unless~ 0             1
## 2 good case, excellent value.                              1             2
## 3 great for the jawbone.                                   1             3
```

### Unusual Characters

In order to reduce the overall character set, we will want to address sentiment-agnostic characters like punctuation, emojis, non-English characters, etc.

**Before:** We see below that the usage of special characters varies widely: punctuation, typos, and emjois.

```
set.seed(213)
subset[sample(nrow(subset), 10), 1]
```

```
## # A tibble: 10 x 1
##    Review
##    <chr>
##  1 never got it!!!!!
##  2 it was horrible!.
##  3 authentic leather with nice shine and comfort .i recommend you this case !!
##  4 we loved the biscuits!!!
##  5 all i have to say is the food was amazing!!!
##  6 the best phone in market :).
##  7 the chips and sals a here is amazing!!!!!!!!!!!!!!!!!!
##  8 for people who are first timers in film making, i think they did an excellen~
##  9 my experience was terrible..... this was my fourth bluetooth headset, and wh~
## 10 lot of holes in the plot: there's nothing about how he became the emperor; n~
```

**Conversion:** We will remove all unwanted unusual characters to reduce the extraneous portions of the vector space by replacing those characters with spaces.

```
df$Review <- gsub("[^a-zA-Z0-9]", " ", df$Review)
```

**After:** Now, we see below that all the special characters have been removed.

```
head(df[grep("[^a-zA-Z0-9]", df$Review), ], n=10)
```

```
## # A tibble: 10 x 3
##    Review                                                   Sentiment Index
##    <chr>                                                    <fct>     <int>
##  1 "so there is no way for me to plug it in here in the us unle~ 0            1
##  2 "good case  excellent value "                            1             2
##  3 "great for the jawbone "                                 1             3
##  4 "tied to charger for conversations lasting more than 45 minu~ 0            4
##  5 "the mic is great "                                      1             5
##  6 "i have to jiggle the plug to get it to line up right to get~ 0            6
##  7 "if you have several dozen or several hundred contacts  then~ 0            7
##  8 "if you are razr owner   you must have this "            1             8
##  9 "needless to say  i wasted my money "                    0             9
## 10 "what a waste of money and time  "                       0            10
```

### Vocabulary Size

We will want to know how many unique words are used within the body of the reviews that we will be analyzing. This will be important when defining the tokenization process as we may elect to only consider some percentage or total words according to how frequently they appear. To begin, however, we will consider each and every word used by the reviewers.

```
num_words <- df %>%
              unnest_tokens(word, Review) %>%
              count(word) %>%
              nrow()

num_words
```

```
## [1] 5183
```

We see that our current vocabulary size is 5183. We will save this value for use during the creation of our text vectorization layer (tokenization).

### Word Embedding Length

In this context, word embedding is what will allow us to generate an efficient and dense representation of the words used by reviewers for use by the machine learning algorithms that we will employ in this NLP neural network.

While some experimentation may be required to optimize the word embedding length, we will begin by using a common guideline that takes the fourth root of the number of words in our vocabulary (Allaire, 2020).

```
num_words ** .25
```

```
## [1] 8.484872
```

In order to ensure that we capture sufficient detail in how the words relate to each other, we will round this value up to 10 and adjust if needed.

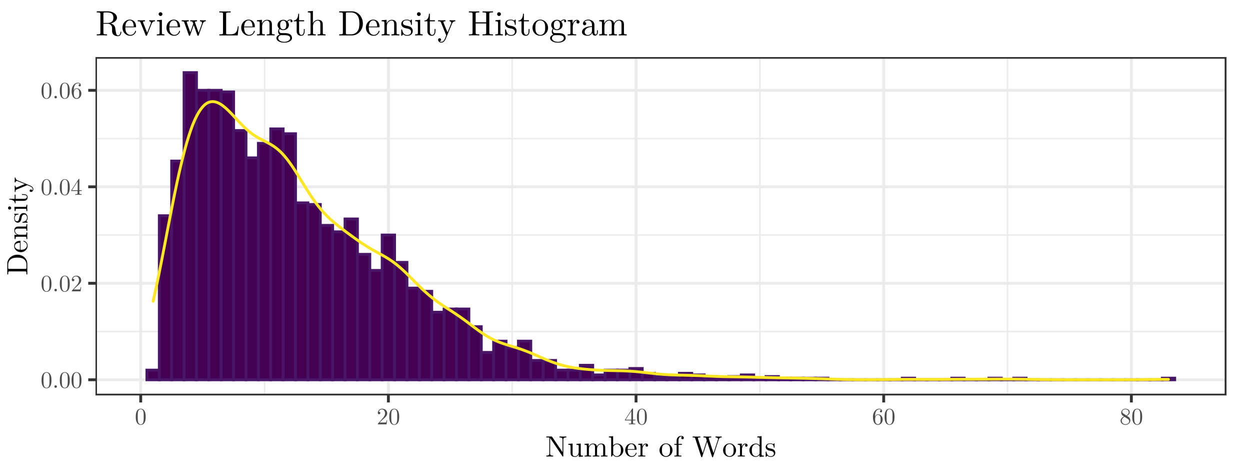
**Maximum Sequence Length**

We see the summary statistics of the review length below. The average review length is approximately 13 words, with a review length range of 1 to 83 words. We will save the length of our longest review here so that we know how much we need to pad shorter reviews to standardize the review length for analysis.

```
a  <- df$Review %>%
  strsplit(" ") %>%
  sapply(length)

max_length <- max(a)

summary(a)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    6.00   11.00   12.99   18.00   83.00
```

While the summary statistics give us a good bit of information, we will also include a histogram to better visualize the distribution of the lengths of these reviews.



## B2: Tokenization

The tokenization process breaks down an element of our dataset (the body of review text) into smaller parts to allow a more granular analysis. This process will allow the words inside of the reviews in our dataset to be analyzed both individually and within the context of the reviews in which they appear. We will perform this step concurrently with the padding process below as part of the text vectorization layer creation.

## B3: Padding

Rather than one-hot encoding our text to $n$-dimensional vectors, we will coerce each of the arrays into having the same length regardless of the number of characters in the review itself. The additions will be blank characters, so they will not affect the sentiment predictions. Instead they will be spaces added on to the end of the original text sequences.

We will use the previously saved variables *num_words* and *max_length* to define the maximum number of tokens as the number of words in the vocabulary of our dataset and the padded length of our observations to the longest of the reviews. This will ensure that (at least initially) we keep all available data (Allaire, 2019).

```
text_vectorization <- layer_text_vectorization(
  max_tokens = num_words,
  output_sequence_length = max_length)
```

```
## Loaded Tensorflow version 2.9.1
```

```
text_vectorization %>%
  adapt(df$Review)
```

This process has essentially transformed each review into a vector where each word is represented by an integer value based on how common the word is within our dataset vocabulary. Below, we will show the original text of one review along with its vectorized (tokenized) and padded form for comparison.

### Original Review

```
df$Review[88]
```

```
## [1] "the construction of the headsets is poor "
```

### Padded and Vectorized Review

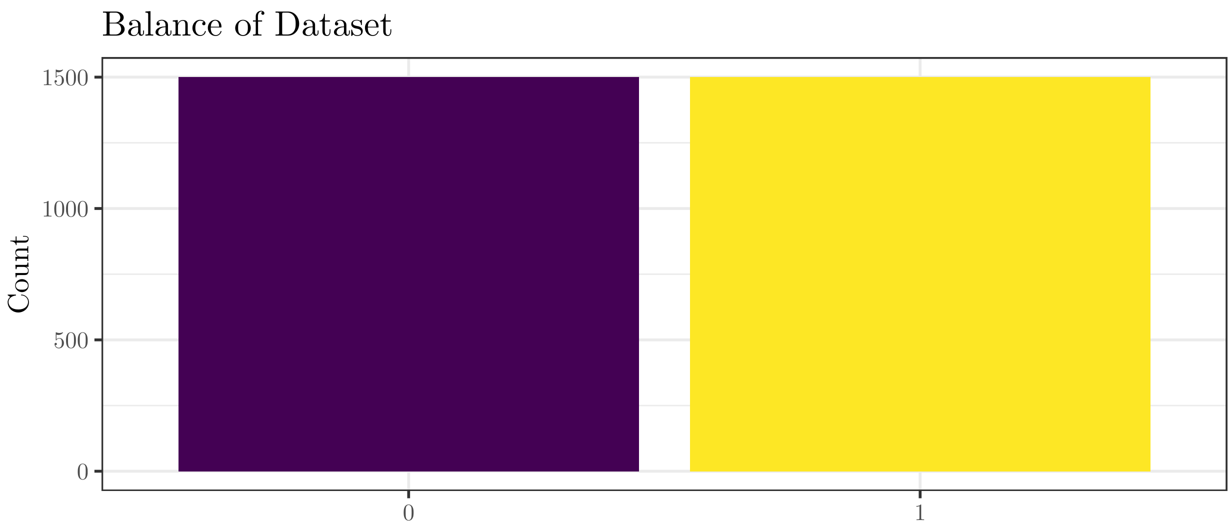
```
text_vectorization(matrix(df$Review[88], ncol = 1))
```

```
## tf.Tensor(
## [[   2 1418   10    2  626    7  179    0    0    0    0    0    0    0
##      0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      0    0    0    0    0    0    0    0    0    0    0    0    0]], shape=(1, 83), dtype=int64)
```

## B4: Sentiment Categories

Our selected dataset has two categories: positive and negative. According to the data dictionary that accompanied the raw data, the sentiment values are coded numerically such that 1 indicates a positive sentiment and 0 indicates a negative sentiment (Kotzias et al, 2015). The sentiment categories are evenly divided to reduce the potential for sample bias.

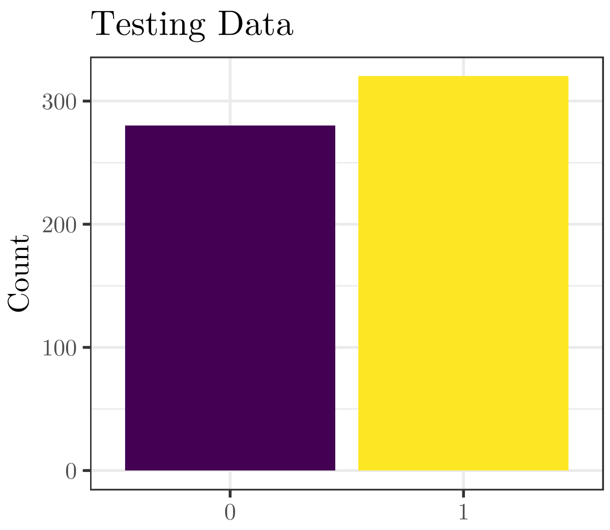
```
##    0    1
## 1500 1500
```

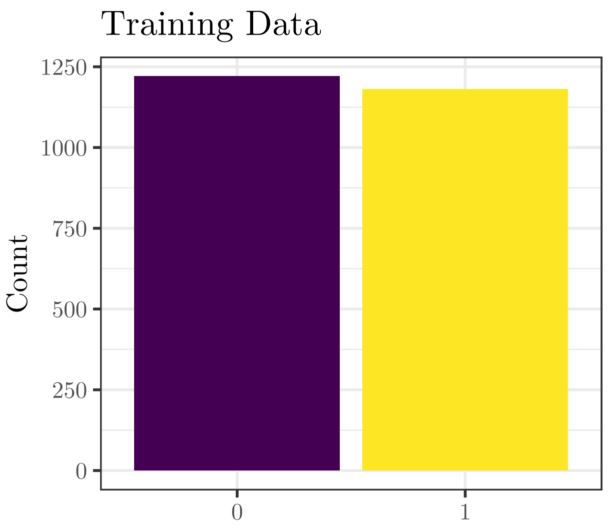


### B5: Train/Test Split

We will adhere to a fairly common 80/20 train/test split for the purposes of this project.

```
set.seed(213)
split.index <- sample.int(nrow(df), size = nrow(df)*0.8)
df.train <- df[split.index,]
df.test <- df[-split.index,]
```





### B6: Data Export

We will save the IMDb review data as a training set and a testing set as divided above using a pseudo-random sample function.

```
write.csv(df.train, './nn_testing.csv', row.names = FALSE)
write.csv(df.test, './nn_training.csv', row.names = FALSE)
```

## Part III: Network Architecture

In this section, we will create the model itself, provide its summary, and then discuss each of its relevant components. The layers of our model are essentially stacked one on top of another in an arranged sequence. Each layer takes the output of the former and passes on to the next. Below, we arrange and define our neural network model.

```
input <- layer_input(shape = c(1), dtype = "string")

output <- input %>%
  text_vectorization() %>%
  layer_embedding(input_dim = num_words + 1, output_dim = 10) %>%
  layer_global_average_pooling_1d() %>%
  layer_dense(units = 20, activation = "relu") %>%
  layer_dropout(0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")

model <- keras_model(input, output)
```

### C1: Model Summary

While it is possible to infer most of the below from how we've defined our model and the associated parameters, we will display the model summary here for reference.

```
summary(model)
```

```
## Model: "model"
## ________________________________________________________________________________
##  Layer (type)                Output Shape              Param #    Trainable
## ================================================================================
##  input_1 (InputLayer)        [(None, 1)]               0          Y
##  text_vectorization (TextVecto  (None, 83)             0          Y
##  rization)
##  embedding (Embedding)       (None, 83, 10)            51840      Y
##  global_average_pooling1d (Glo  (None, 10)             0          Y
##  balAveragePooling1D)
##  dense_1 (Dense)             (None, 20)                220        Y
##  dropout (Dropout)           (None, 20)                0          Y
##  dense (Dense)               (None, 1)                 21         Y
## ================================================================================
## Total params: 52,081
## Trainable params: 52,081
## Non-trainable params: 0
## ________________________________________________________________________________
```

## C2: Layers and Parameters

We are employing a total of seven layers in this model. We will address each of these individually here:

**1. Input Layer:** We have constructed our model such that the input for the model is the cleaned text from our sample reviews. This layer prepares our model for that input.

**2. Text Vectorization:** Because we are not providing already-tokenized data to the model, we will take care of that step here using the parameters that we previously defined.

**3. Embedding Layer:** Our embedding layer receives vectorized arrays from the text vectorization layer and then looks up the embedding vector for each word in the associated review according to how it is indexed.

We note that there are 51,840 parameters here. This is a result of our vocabulary size being 5183 and our determined initial word embedding length of 10.

**4. Global Average Pooling** This layer will give a fixed length vector as an output for each review which will enable our model to process each review embedding output in a similar way, regardless of their length.

**5. Dense Layer** This is a simple dense (fully-connected) layer with 20 nodes. Their behavior will be trained according to their inputs. The 220 parameters here is a result of the 10 input values and 20 nodes.

**6. Dropout** We incorporate a dropout layer here to help prevent overfitting.

**7. Dense Layer** The last layer here is a dense (fully-connected) single output node to perform the final classification. The output value will be a float between 0 and 1 (as a result of the sigmoid activation function) and will represent a probability for classification purposes. The 21 parameters are a result of the 20 input values and its single node.

## C3: Hyperparameters

In order to create our model, we must first make some decisions about model inputs that the model does not self-generate. These decisions concern functions and values are dependent on the use case of the model or certain insights an analyst may have regarding a project. The hyperparameter decision details for this project will be explored in more detail here.

### Activation Functions

We use two distinct activation functions in our model: one in each of our dense layers.

For our first dense layer (layer five), we use the Rectified Linear Unit (ReLU) activation function. In most use cases, ReLU runs more quickly than alternatives and generally produces usable results with minimal fine tuning (Bag, 2021).

For our final layer, we use the Sigmoid activation function. The range of the output of a Sigmoid function is $[0, 1]$, which is the same range that we use to express probability. A Sigmoid activation function is an ideal choice when a binary classification is the desired output (Bag, 2021).

### Number of Nodes per Layer

With respect to defining the number of nodes per layer as a hyperparameter, we will address our embedding layer, our first dense layer, and our second (final) dense layer.

**Embedding Layer Nodes:** Here, the number of nodes is dictated by the number of words in our dataset vocabulary. This allows our model to ingest the input values of our reviews.

**Dense Layer 1:** The number of nodes in this layer must be less than the number of nodes in the Embedding Layer but more than the number of nodes in our final Dense layer. In our case, the range of possible values is $(1, 5183)$. Our selection can consider computational cost implications and risk of overfitting, but there is an arbitrary element to this decision. We selected 20 nodes as a compromise between sufficient complexity to produce an accurate model, not too complex to impose computational limitations, and not so high as to guarantee too tight a fit to the training data.

**Dense Layer 2:** Here, our desired output is a single probability value to yield a binary classification. For this reason, a single output node is the natural choice.

### Loss Function

The final output of our model is a binary classification of the input reviews as either negative or positive in tone. This means our final output will be a probability; this is why the last layer of our model consists of a single node with a Sigmoid activation function. Under these circumstances, the *binary_crossentropy* loss function is ideal, and that is what we will use in Part IV..

### Optimizer

In Part IV, we will be using the Adam optimizer. Adam optimization takes the principle advantages of alternative optimizer options (e.g. stochastic gradient descent) and reduces their computational requirements and minimizes memory usage despite assigning individual adaptive learning rates to different parameters. Adam is appropriate for use in many applications (including classification) and yields equivalent if not better results than more computationally expensive alternatives (Kingma & Ba, 2014).

### Stopping Criteria

Invoking an early stopping criteria is a good method for preventing overfitting. It monitors the performance of the model on an epoch-by-epoch basis using a held-back subset of the training data, and it stops the model from continuing to fit if the validation quality ceases to improve. We will determine whether early stopping is necessary in Part IV, and–if it is required–we will apply a *val_accuracy* early stopping function from the Keras module.

### Evaluation Metric

When assessing the quality of our model, we will rely on two main evaluation metrics: (validation) accuracy and loss.

**(Validation) Accuracy:** The simplest metric to interpret and present to stakeholders is accuracy. It is, quite intuitively, the ratio of correct predictions to total predictions. Given that an model that accurately predicts the sentiment of a text is our primary goal, it follows that accuracy will be our primary metric of concern. However, we will specifically focus on validation accuracy as this will examine how well our model predicts the sentiment of held back training data. We prioritize validation accuracy over simple accuracy to avoid overfitting.

**Loss:** Loss is the error (given by our loss function) that (inversely to accuracy) we want to minimize. We expect loss to decrease as accuracy increases; however, a loss function may not necessarily allow us to infer our model's accuracy. For this reason, we will examine loss and accuracy together to try to determine the quality of our model's behavior.

## Part IV: Model Evaluation

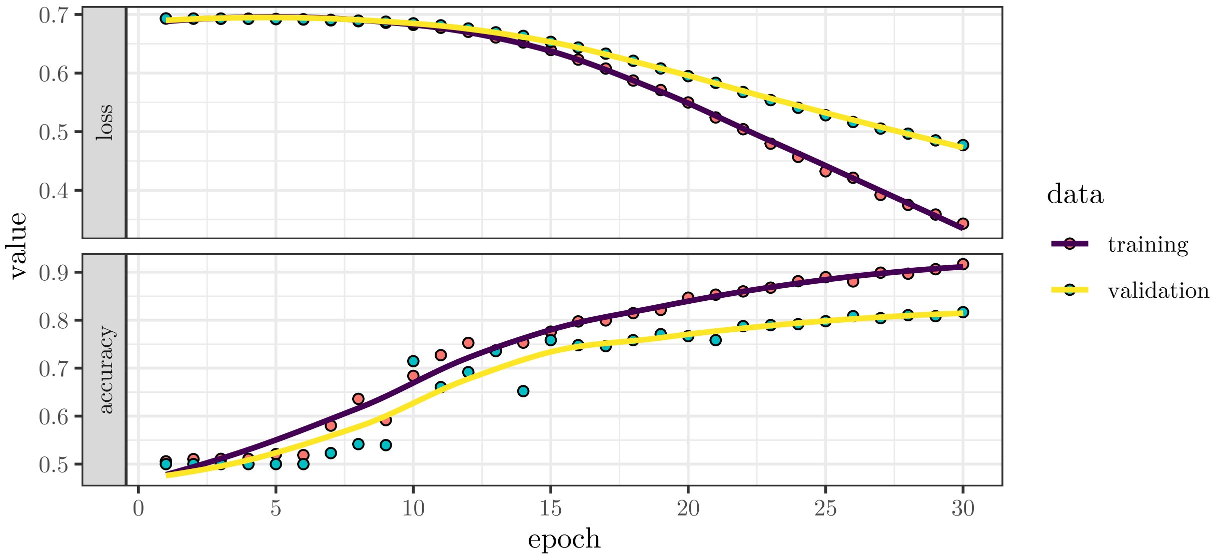
With the network layers and parameters created as discussed above, we will now train and evaluate the model using the dataset that we prepared above in Part II.

```r
model %>% compile(optimizer = 'adam',
                  loss = 'binary_crossentropy',
                  metrics = list('accuracy'))

history.1 <- model %>% fit(
            df.train$Review,
            as.numeric(df.train$Sentiment == 1),
            epochs = 30,
            batch_size = 100,
            validation_split = 0.2,
            verbose=0)
history.1
```

```
##
## Final epoch (plot to see history):
##         loss: 0.343
##     accuracy: 0.9167
##     val_loss: 0.4771
## val_accuracy: 0.8167
```

We see the validation accuracy is pretty good at 0.8167, but we see that the loss is higher than we would like at 0.343. We will make some adjustments to our neural network parameters to see if we can get a more even result than this. In general, we would like to see a much lower loss and a much higher validation accuracy.

We will also examine the history plot to determine whether we should implement early stopping.



There does seem to be a widening between the training and validation metrics that is sufficiently troubling that we will add early stopping to our model training as well.

### Model Tuning

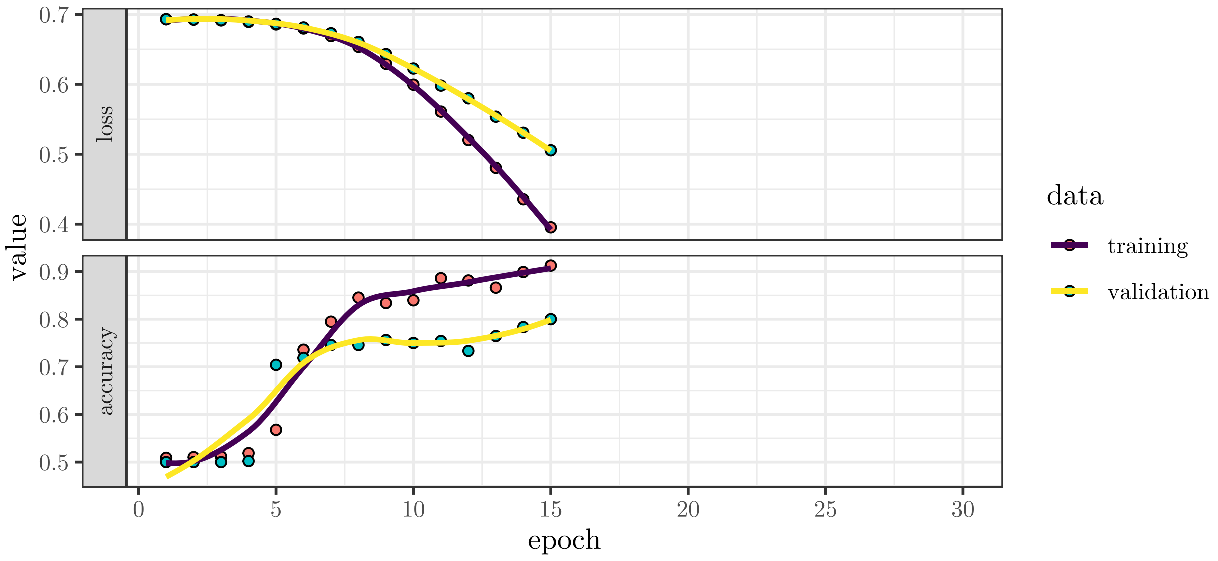
With the insights from our initial attempt, we have decided to make our neural network slightly more complex by increasing the number of nodes in two places. We have also slightly reduced the dropout rate because we added early stopping as another countermeasure to overfitting. Finally, we adding an early stopping function to ensure that we maximize the quality of our model without overfitting.

```
output <- input %>%
  text_vectorization() %>%
  layer_embedding(input_dim = num_words + 1, output_dim = 30) %>% # Increased Nodes
  layer_global_average_pooling_1d() %>%
  layer_dense(units = 50, activation = "relu") %>%                # Increased Nodes
  layer_dropout(0.2) %>%                                          # Reduced Drop Rate
  layer_dense(units = 1, activation = "sigmoid")

callbacks <- callback_early_stopping(                             # Define Stop Early
  monitor = "val_accuracy",
  min_delta = .1,
  patience = 10,
  verbose = 1,
  mode = "max")

model <- keras_model(input, output)

model %>% compile(optimizer = 'adam',
                  loss = 'binary_crossentropy',
                  metrics = list('accuracy'))

history.2 <- model %>% fit(
  df.train$Review,
  as.numeric(df.train$Sentiment == 1),
  epochs = 30,
  batch_size = 100,
  validation_split = 0.2,
  verbose=0,
  callbacks=callbacks)                                           # Add Stop Early

history.2
```

```
##
## Final epoch (plot to see history):
##         loss: 0.3955
##     accuracy: 0.9125
##     val_loss: 0.5056
## val_accuracy: 0.8
```

The results of this model are considerably better. We see the validation accuracy is slightly better at 0.8, but now we also see that the loss is much more in line with the low values we want from a quality model at 0.3954636.

Again, we will plot the history chart to see how the early stopping measure preventing us from overfitting.



## D1: Final Epoch

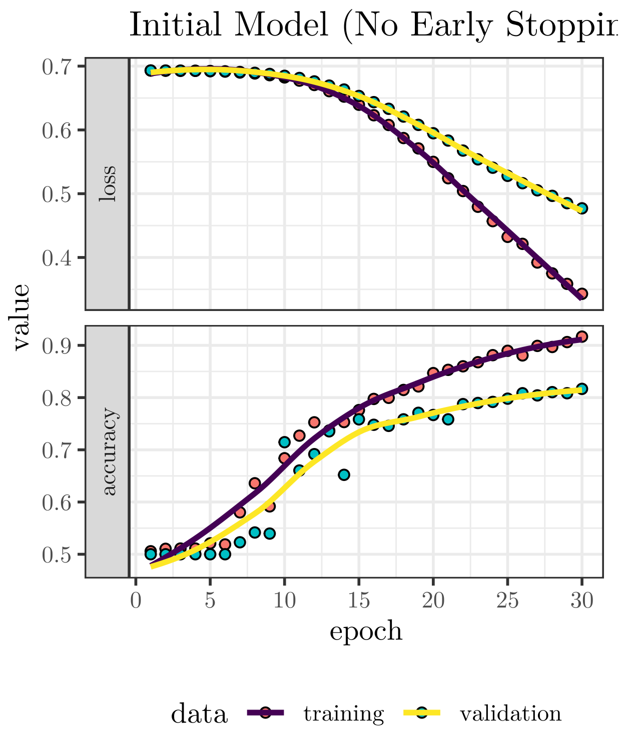
As discussed above, our first model had good accuracy metrics but poor loss values. After making some adjustments and accounting for early stopping, we see a very low loss and a very high accuracy. This is a much better optimized model than our initial attempt.

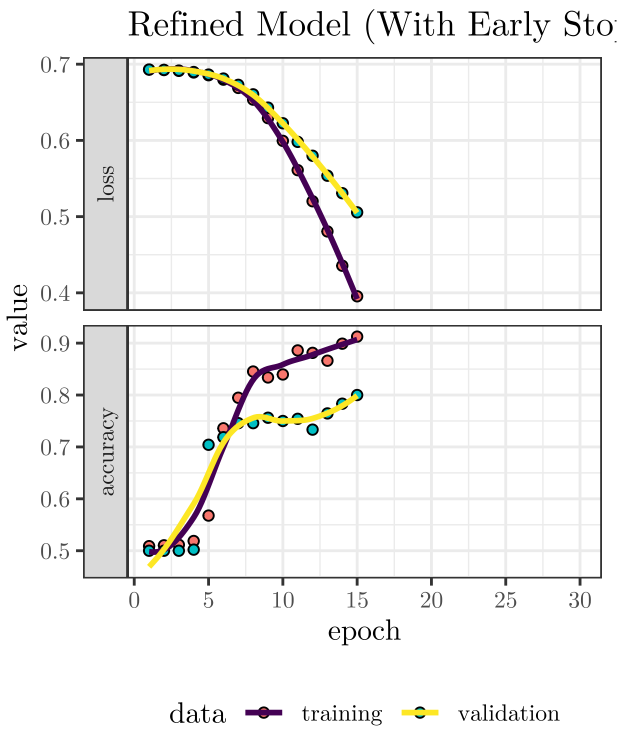
```
history.2
```

```
##
## Final epoch (plot to see history):
##         loss: 0.3955
##     accuracy: 0.9125
##     val_loss: 0.5056
## val_accuracy: 0.8
```

## D2: Training Progress

The loss and accuracy metrics have been a significant area of focus for us during the course of training our model. These metrics indicate how well our model predicts unseen values, give an idea of how well our results will generalize, and could suggest shortcomings that allow us to improve our results. Below we will show visualizations for our loss and accuracy metrics by epoch for a side-by-side comparison.





### D3: Model Fitness

We should not be too concerned if our model fits training data more closely than validation data; however, we want to ensure that gains in training accuracy are accompanied by a correlating increase in validation accuracy. We have taken several measures to ensure a quality fit that generalizes well.

There is a gap between the training and validation metrics; by definition, there is some slight overfitting to this model—even the final version of it. We want to highlight that we included a dropout layer in our model to reduce the effects of overfitting. Also, we incorporated an early stop to cease model fitting when the validation accuracy was not increasing sufficiently with each epoch. These measures helped to minimize the metrics gap while allowing the generalized validation accuracy to optimize.

### D4: Predictive Accuracy

To this point, we have only utilized the training data to tune and assess our model. While our methods utilized held-back values for validation purposes, we will want to see how well our model predicts the sentiments of unseen reviews.

```r
results <- model %>% evaluate(df.test$Review, as.numeric(df.test$Sentiment == 1), verbose = 0)
results
```

```
##      loss  accuracy
## 0.5094700 0.8066667
```

In this case, when dealing with unseen data, our primary focus will be on accuracy. We will be able to approach this projects stakeholders with a model that provides accurate predictions 80.67% of the time.

# Part V: Summary and Recommendations

## E: Save Trained Model

The code segments shown here pertain to the saving of our trained neural network.

```
model %>% save_model_tf("NLP_Trained")
```

## F: Code Functionality

The end result of our project indicates that the functionality of our neural network is such that we can apply it to business needs. The computational requirements are not such that significant resources are required, and this is a direct consequence of our selected network architecture. We chose activation functions, loss functions, and an optimizer that work well with each other and produce optimal results with efficiency.

Some other options would require a significant increase in available memory, a recommended dedicated GPU, and—even then—additional time requirements. Such an approach is not suitable to our specific business ends. We have, as directed, chosen an appropriate path for this proof of concept project.

## G: Recommendations

Despite the "black box" nature of neural networks, we have successfully demonstrated the requested proof of concept as directed by hospital administrators. We are able to access writer sentiment using an NLP model with a sufficient accuracy (80.67%) to recommend that the hospital allow free response sections to their surveys. This will allow the stakeholders to have access to metrics beyond simple patient prioritization of services and open up new avenues of approach to improve hospital key performance indicators and improve patient outcomes.

---

# Part VI: Reporting

---

## H: IDE Selection

The entirety of this submission was authored within an R Notebook using RStudio Server version 2022.02.2, build number 485 (RStudio Team, 2022). This instance was installed on a virtual instantiation of Ubuntu Server 20.04.4 LTS.

As we have done with all other submissions in this program, we will include a knitted pdf of this executed notebook.

## I-J: Academic Integrity

All sources for research and code usage have been cited in-text where appropriate, and a complete listing of these references is provided below.

## K: Communication Standards

This content has been edited for professionalism and appropriate communication standards. The format of this submission adheres to rubric requirements to the best of the author's ability, and any deviations will be corrected as necessary.

## References

Allaire, J. (2019). Basic Text Classification. The TensorFlow Authors and RStudio, PBC. Retrieved July 4, 2022, from https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_text_classification/

Allaire, J. (2020). Feature Columns. The TensorFlow Authors and RStudio, PBC. Retrieved July 4, 2022, from https://tensorflow.rstudio.com/guide/tfdatasets/feature_columns/

Bag, S. (2021, April 26). Activation functions - all you need to know! Medium. Retrieved July 4, 2022, from https://medium.com/analytics-vidhya/activation-functions-all-you-need-to-know-355a850d025e

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kotzias, D., Denil, M., Freitas, N. & Smyth, P. 2015. From Group to Individual Labels Using Deep Features. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15). Association for Computing Machinery, New York, NY, USA, 597–606. https://doi.org/10.1145/2783258.2783380

RStudio Team (2022). RStudio: Integrated Development Environment for R. RStudio, PBC, Boston, MA. http://www.rstudio.com/.

Saxena, S. (2021, February 6). Understanding embedding layer in Keras. Medium. Retrieved July 4, 2022, from https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce