Deep Learning
for
Text Classification

By
Omkar Nandkumar Dixit (ond170030)
Karan Yashwant Kanani(kyk170030)

What is Text Classification?

Text classification is the process of assigning tags or categories to **text** according to its content. It's one of the fundamental tasks in Natural Language Processing (NLP) with broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection.

We have taken the SMS Spam Collection Data Setin which we have to mark SMS as spam or not spam(legit). In this report and in the code the words legit SMS means not spam.

We have used the following libraries to achieve the result.

```
library(keras)
library(cloudml)
library(SnowballC)
library(wordcloud)
library(stringr)
library(tm)
```

Let's start with reading the data

```
rawInputData <-
read.csv("http://www.utdallas.edu/~ond170030/data/SMSSpamCollection.csv",
stringsAsFactors = FALSE)
```

Changing the names of the columns, for some weird reasons we were getting extra columns, so we decided to take the subset to make sure we are not fed extra data.

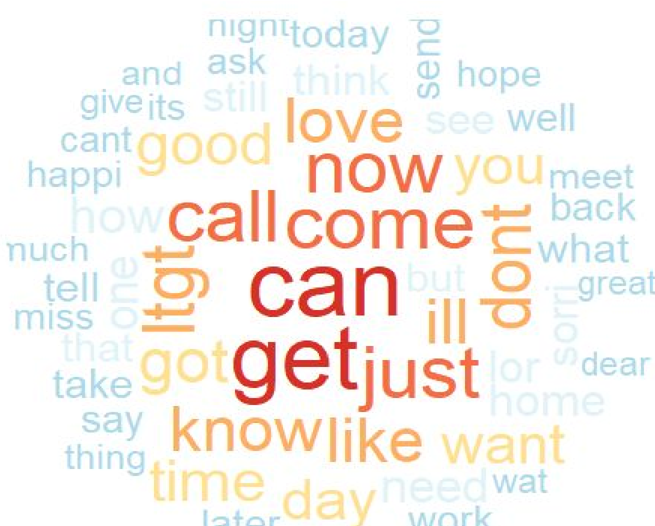
```
colnames(rawInputData) <- c("Type", "Text")
rawInputData <- subset(rawInputData, select = c("Type", "Text"))
```

Lets just see the words that have a very high frequency

```
corpusFull <- VCorpus(VectorSource(rawInputData$Text))
corpusFull <- tm_map(corpusFull, removeNumbers)
corpusFull <- tm_map(corpusFull, removePunctuation)
corpusFull <- tm_map(corpusFull, removeWords, stopwords())
corpusFull <- tm_map(corpusFull, stripWhitespace)
corpusFull <- tm_map(corpusFull, stemDocument)
wordcloud(corpusFull, min.freq = 100, random.order = FALSE, color = (colors =
c("#4575b4","#74add1","#abd9e9","#e0f3f8","#fee090","#fdae61","#f46d43","#d73027")))
```

We get a wordcloud showing the words with a high frequency

We do the same for Legit and Spam respectively



Okay let's get back to what we are aiming to do

We now do Train Test Split for 70% to training and 30% to testing

```
sample.size <- floor(0.70 * nrow(rawInputData))
set.seed(123)
trainIndex <- sample(seq_len(nrow(rawInputData)), size = sample.size)
trainData <- rawInputData[trainIndex, ]
testData <- rawInputData[-trainIndex, ]
```

We have to preprocess Train data using corpus which includes removing numbers, remove Punctuation, remove stop words, strip white spaces, and stemming

```
corpusTrain <- VCorpus(VectorSource(trainData))
corpusTrain <- tm_map(corpusTrain, removeNumbers)
corpusTrain <- tm_map(corpusTrain, removePunctuation)
corpusTrain <- tm_map(corpusTrain, removeWords, stopwords())
corpusTrain <- tm_map(corpusTrain, stripWhitespace)
corpusTrain <- tm_map(corpusTrain, stemDocument)
```

We have to preprocess Train data using corpus which includes removing numbers, remove Punctuation, remove stop words, strip white spaces, and stemming

```
corpusTest <- VCorpus(VectorSource(testData))
corpusTest <- tm_map(corpusTest, removeNumbers)
corpusTest <- tm_map(corpusTest, removePunctuation)
corpusTest <- tm_map(corpusTest, removeWords, stopwords())
corpusTest <- tm_map(corpusTest, stripWhitespace)
corpusTest <- tm_map(corpusTest, stemDocument)
```

We will be extracting the data from the corpus

```
cleanTrainData <- c()
cleanTrainLabels <- c()
for(i in 1:length(corpusTrain[[2]]$content)){
 if(str_length(corpusTrain[[2]]$content[i]) > 1){
  cleanTrainData <- c(cleanTrainData, corpusTrain[[2]]$content[i])
  if(corpusTrain[[1]]$content[i] == "spam")
   cleanTrainLabels<-c(cleanTrainLabels,1)
  else
   cleanTrainLabels<-c(cleanTrainLabels,0)
 }
```

```
}

cleanTestData <- c()
cleanTestLabels <- c()
for(i in 1:length(corpusTest[[2]]$content)){
  if(str_length(corpusTest[[2]]$content[i]) > 1){
    cleanTestData<-c(cleanTestData,corpusTest[[2]]$content[i])
    if(corpusTest[[1]]$content[i] == "spam")
      cleanTestLabels<-c(cleanTestLabels,1)
    else
      cleanTestLabels<-c(cleanTestLabels,0)
  }
}
```

Our tokenizer function

```
tokenizer <- text_tokenizer(num_words = 10000, filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n",
                            lower = TRUE, split = " ", char_level = TRUE, oov_token = NULL)%>%
  fit_text_tokenizer(cleanTrainData)

# Creating Sequential data which will be later vectorzied
cleanTrainDataSeq <- texts_to_sequences(tokenizer, cleanTrainData)
cleanTestDataSeq <- texts_to_sequences(tokenizer, cleanTestData)
```

Our Vectorization function

```
vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences),
                    ncol = dimension)
  for (i in 1:length(sequences))
    results[i, sequences[[i]]] <- 1
  results
}
```

Finally we get our vectorized data from the clean sequential data

```
finalTrainingData <- vectorize_sequences(cleanTrainDataSeq)
finalTestingData <- vectorize_sequences(cleanTestDataSeq)
```

Converting our labels into numeric form

```
finalTrainingLabels <- as.numeric(cleanTrainLabels)
finalTestingLabels <- as.numeric(cleanTestLabels)
```

Our Keras Model

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

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

Fitting the training data

```
model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 4, batch_size = 512)
```

Testing on testing data

```
results <- model %>% evaluate(finalTestingData, finalTestingLabels)
```

We will split data into train and validation parts, to just evaluate our model more

```
sample.size <- floor(0.70 * nrow(finalTrainingData))
set.seed(123)
val_indices <- sample(seq_len(nrow(finalTrainingData)), size = sample.size)
x_val <- finalTrainingData[val_indices,]
partial_x_train <- finalTrainingData[-val_indices,]
y_val <- finalTrainingLabels[val_indices]
partial_y_train <- finalTrainingLabels[-val_indices]

history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
```

Model Runs

Model	Result

model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
            input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
 	
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
 	
model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 4, batch_size = 512)
 	
results <- model %>% evaluate(finalTestingData, finalTestingLabels)
 	
results
	$loss
	[1] 0.411945
	
	$acc
	[1] 0.8581688
	
	Google Cloud Results

model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu",
            input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "tanh")
 	
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
 	
model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 8, batch_size = 512)
	$loss
	[1] 2.286049
	
	$acc
	[1] 0.8581688
	
	Google Cloud Results

results <- model %>% evaluate(finalTestingData, finalTestingLabels) results	

model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 1, activation = "tanh")
 	
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
 	
model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 12, batch_size = 512)
 	
results <- model %>% evaluate(finalTestingData, finalTestingLabels)
 	
results
	$loss
	[1] 2.286049
	
	$acc
	[1] 0.8581688
	
	Google Cloud Results

model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "relu")
 	
model %>% compile(
  optimizer = "rmsprop",
  loss = "poisson",
  metrics = c("accuracy")
)
 	
model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 8, batch_size = 512)
 	
results <- model %>% evaluate(finalTestingData, finalTestingLabels)
 	
results
	$loss
	[1] 0.2835351
	
	$acc
	[1] 0.9054458
	
	Google Cloud Results

model <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>% layer_dense(units = 16, activation = "relu") %>% layer_dense(units = 32, activation = "relu") %>% layer_dense(units = 16, activation = "relu") %>% layer_dense(units = 16, activation = "relu") %>% layer_dense(units = 1, activation = "relu") model %>% compile(optimizer = "rmsprop", loss = "poisson", metrics = c("accuracy")) model %>% fit(finalTrainingData, finalTrainingLabels, epochs = 10, batch_size = 512) results <- model %>% evaluate(finalTestingData, finalTestingLabels) results	$loss [1] 0.2785691 $acc [1] 0.904249 Google Cloud Results