Amazon Reviews
Sentiment Analysis/Text Classification
Choose Your Own Project
A Harvard Capstone Project

Manoj Bijoor

July 28, 2021

Abstract

Deriving truth and insight from a pile of data is a powerful but error-prone job.

This project offers an empirical exploration on the use of Neural networks for text classification using the Amazon Reviews Polarity dataset.

Text classification algorithms are at the heart of a variety of software systems that process text data at scale.

One common type of text classification is sentiment analysis, whose goal is to identify the polarity of text content: the type of opinion it expresses. This can take the form of a binary like/dislike rating, or a more granular set of options, such as a star rating from 1 to 5. Examples of sentiment analysis include analyzing Twitter posts to determine if people liked the Black Panther movie, or extrapolating the general public's opinion of a new brand of Nike shoes from Walmart reviews.

Algorithms such as regularized linear models, support vector machines, and naive Bayes models are used to predict outcomes from predictors including text data. These algorithms use a shallow (single) mapping. In contrast, Deep learning models approach the same tasks and have the same goals, but the algorithms involved are different. Deep learning models are "deep" in the sense that they use multiple layers to learn how to map from input features to output outcomes.

Deep learning models can be effective for text prediction problems because they use these multiple layers to capture complex relationships in language.

The layers in a deep learning model are connected in a network and these models are called Neural Networks.

Neural language models (or continuous space language models) use continuous representations or embeddings of words to make their predictions. These models make use of Neural networks.

Continuous space embeddings help to alleviate the curse of dimensionality in language modeling: as language models are trained on larger and larger texts, the number of unique words (the vocabulary) increases. The number of possible sequences of words increases exponentially with the size of the vocabulary, causing a data sparsity problem because of the exponentially many sequences. Thus, statistics are needed to properly estimate probabilities. Neural networks avoid this problem by representing words in a distributed way, as non-linear combinations of weights in a neural net.

Instead of using neural net language models to produce actual probabilities, it is common to instead use the distributed representation encoded in the networks' "hidden" layers as representations of words; each word is then mapped onto an n-dimensional real vector called the word embedding, where n is the size of the layer just before the output layer. An alternate description is that a neural net approximates the language function and models semantic relations between words as linear combinations, capturing a form of compositionality.

In this project we will cover four network architectures, namely DNN, CNN, sepCNN and BERT. We will also first implement a Baseline linear classifier model which serves the purpose of comparison with the deep learning techniques.

For metrics we will use the default performance parameters for binary classification which are Accuracy, Loss and ROC AUC (area under the receiver operator characteristic curve).

Contents

List of tables

List of figures

List of Equations

1 Project Overview: Amazon Reviews Polarity

1.1 Introduction

Deriving truth and insight from a pile of data is a powerful but error-prone job.

Text classification algorithms are at the heart of a variety of software systems that process text data at scale.

One common type of text classification is sentiment analysis, whose goal is to identify the polarity of text content: the type of opinion it expresses. This can take the form of a binary like/dislike rating, or a more granular set of options, such as a star rating from 1 to 5. Examples of sentiment analysis include analyzing Twitter posts to determine if people liked the Black Panther movie, or extrapolating the general public's opinion of a new brand of Nike shoes from Walmart reviews.

Algorithms such as regularized linear models, support vector machines, and naive Bayes models are used to predict outcomes from predictors including text data. These algorithms use a shallow (single) mapping. In contrast, Deep learning models approach the same tasks and have the same goals, but the algorithms involved are different. Deep learning models are "deep" in the sense that they use multiple layers to learn how to map from input features to output outcomes.

Deep learning models can be effective for text prediction problems because they use these multiple layers to capture complex relationships in language.

The layers in a deep learning model are connected in a network and these models are called neural networks.

1.1.1 Neural networks

Neural language models (or continuous space language models) use continuous representations or embeddings of words[1] to make their predictions.Karpathy, Andrej. "The Unreasonable Effectiveness of Recurrent Neural Networks"[2] These models make use of Neural networks[3].

Continuous space embeddings help to alleviate the curse of dimensionality[4] in language modeling: as language models are trained on larger and larger texts, the number of unique words (the vocabulary) increases.Heaps' law[5]. The number of possible sequences of words increases exponentially with the size of the vocabulary, causing a data sparsity problem because of the exponentially many sequences. Thus, statistics are needed to properly estimate probabilities. Neural networks avoid this problem by representing words in a distributed way, as non-linear combinations of weights in a neural net.Bengio, Yoshua (2008). "Neural net language models". Scholarpedia. 3. p. 3881. Bibcode:2008SchpJ...3.3881B. doi:10.4249/scholarpedia.3881[6] An alternate description is that a neural net approximates the language function.

Instead of using neural net language models to produce actual probabilities, it is common to instead use the distributed representation encoded in the networks' "hidden" layers as representations of words;
A hidden layer is a synthetic layer in a neural network between the input layer (that is, the features) and the output layer (the prediction). Hidden layers typically contain an activation function such as ReLU[7] for training. A deep neural network contains more than one hidden layer. Each word is then mapped onto an n-dimensional real vector called the word embedding, where n is the size of the layer just before the output layer. The representations in skip-gram models for example have the distinct characteristic that they model semantic relations between words as linear combinations[8], capturing a form of compositionality[9].

[1] https://en.wikipedia.org/wiki/Word_embedding
[2] https://karpathy.github.io/2015/05/21/rnn-effectiveness/
[3] https://en.wikipedia.org/wiki/Artificial_neural_network
[4] https://en.wikipedia.org/wiki/Curse_of_dimensionality
[5] https://en.wikipedia.org/wiki/Heaps%27_law
[6] https://ui.adsabs.harvard.edu/abs/2008SchpJ...3.3881B/abstract
[7] https://developers.google.com/machine-learning/glossary?utm_source=DevSite&utm_campaign=Text-Class-Guide&utm_medium=referral&utm_content=glossary&utm_term=sepCNN#rectified-linear-unit-relu
[8] https://en.wikipedia.org/wiki/Linear_combination
[9] https://en.wikipedia.org/wiki/Principle_of_compositionality

In this project we will cover four network architectures, namely:

1. DNN - Dense Neural Network - a bridge between the "shallow" learning approaches and the other 3 - CNN, sepCNN, BERT.
2. CNN - Convolutional Neural Network - advanced architecture appropriate for text data because they can capture specific local patterns.
3. sepCNN - Depthwise Separable Convolutional Neural Network.
4. BERT - Bidirectional Encoder Representations from Transformers.

We will also first implement a Baseline linear classifier model which serves the purpose of comparison with the deep learning techniques we will implement later on, and also as a succinct summary of a basic supervised machine learning analysis for text.

This linear baseline is a regularized linear model trained on the same data set, using tf-idf weights and 5000 tokens.

For metrics we will use the default performance parameters for binary classification which are Accuracy, Loss and ROC AUC (area under the receiver operator characteristic curve).

We will also use the confusion matrix to get an overview of our model performance, as it includes rich information.

We will use tidymodels packages along with Tensorflow, the R interface to Keras. See Allaire, JJ, and François Chollet. 2021. keras: R Interface to 'Keras'[10] for preprocessing, modeling, and evaluation, and Silge, Julia, and David Robinson. 2017. Text Mining with R: A Tidy Approach. 1st ed. O'Reilly Media, Inc.[11], Supervised Machine Learning for Text Analysis in R, by Emil Hvitfeldt and Julia Silge.[12] and Tidy Modeling with R, Max Kuhn and Julia Silge, Version 0.0.1.9010, 2021-07-19[13] and how can we forget Introduction to Data Science, Data Analysis and Prediction Algorithms with R - Rafael A. Irizarry, 2021-07-03[14].

The keras R package provides an interface for R users to Keras, a high-level API for building neural networks.

This project will use some key machine learning best practices for solving text classification problems. Here's what you'll learn:

1. The high-level, end-to-end workflow for solving text classification problems using machine learning
2. How to choose the right model for your text classification problem
3. How to implement your model of choice using TensorFlow with Keras acting as an interface for the TensorFlow library

I have used/mentioned several references throughout the project.

This project depends on python and R software for Tensorflow and Keras that needs to be installed both inside and outside of R. As each individual's environment may be different, I cannot automate this part in my code.

R side:
https://cran.r-project.org/
https://tensorflow.rstudio.com/installation/
https://tensorflow.rstudio.com/installation/gpu/local_gpu/

[10] https://CRAN.R-project.org/package=keras
[11] https://www.tidytextmining.com/
[12] https://smltar.com/
[13] https://www.tmwr.org/
[14] https://rafalab.github.io/dsbook/

Python side:
https://www.tensorflow.org/install
https://www.anaconda.com/products/individual
https://keras.io/

Instead of cluttering code with comments, I ask you to please use these references and the rstudio help (?cmd/??cmd) if you are not very familiar with any specific command. Most commands are pretty self explanatory if you are even a little familiar with R.

Here are some more references:

1.2 References

Tensorflow[15] is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

The TensorFlow Hub[16] lets you search and discover hundreds of trained, ready-to-deploy machine learning models in one place.

Tensorflow for R[17] provides an R interface for Tensorflow.

Tidy Modeling with R[18]

Tinytex[19]
I have used tinytex in code chunks.

Latex[20]
I have used Latex beyond the very basic provided by default templates in RStudio. Too numerous to explain. Though that much is not needed, I have used it to learn and make better pdf docs.

Rmarkdown[21]

[15] https://www.tensorflow.org/
[16] https://tfhub.dev/
[17] https://tensorflow.rstudio.com/
[18] https://www.tmwr.org/
[19] https://yihui.org/tinytex/
[20] https://www.overleaf.com/learn/latex
[21] https://bookdown.org/yihui/rmarkdown

1.3 Text Classification Workflow

Here's a high-level overview of the workflow used to solve machine learning problems:

Step 1: Gather Data
Step 2: Explore Your Data
Step 2.5: Choose a Model*
Step 3: Prepare Your Data
Step 4: Build, Train, and Evaluate Your Model
Step 5: Tune Hyperparameters
Step 6: Deploy Your Model

The following sections explain each step in detail, and how to implement them for text data.

1.3.1 Gather Data

Gathering data is the most important step in solving any supervised machine learning problem. Your text classifier can only be as good as the dataset it is built from.

Here are some important things to remember when collecting data:

1. If you are using a public API, understand the limitations of the API before using them. For example, some APIs set a limit on the rate at which you can make queries.
2. The more training examples/samples you have, the better. This will help your model generalize better.
3. Make sure the number of samples for every class or topic is not overly imbalanced. That is, you should have comparable number of samples in each class.
4. Make sure that your samples adequately cover the space of possible inputs, not only the common cases.

This dataset contains amazon reviews posted by people on the Amazon website, and is a classic example of a sentiment analysis problem.

Amazon Review Polarity Dataset - Version 3, Updated 09/09/2015

ORIGIN

The Amazon reviews dataset consists of reviews from amazon. The data span a period of 18 years, including ~35 million reviews up to March 2013. Reviews include product and user information, ratings, and a plaintext review. For more information, please refer to the following paper: J. McAuley and J. Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, pages 165–172, New York, NY, USA, 2013. ACM[22].

The Amazon reviews polarity dataset was constructed by Xiang Zhang (xiang.zhang@nyu.edu[23]) from the above dataset. It is used as a text classification benchmark in the following paper: Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification[24]. Advances in Neural Information Processing Systems 28 (NIPS 2015).

Here is an Abstract of that paper:

This article offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. We constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional

[22] https://cs.stanford.edu/people/jure/pubs/reviews-recsys13.pdf
[23] mailto:xiang.zhang@nyu.edu
[24] https://arxiv.org/abs/1509.01626

models such as bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks.

Coming back to our project: As Google has changed it's API, I had to download the dataset manually from the following URL:

Please select file named "amazon_review_polarity_csv.tar.gz" and download it to the project directory.

Download Location URL : Xiang Zhang Google Drive[25]

DESCRIPTION

The Amazon reviews polarity dataset is constructed by taking review score 1 and 2 as negative, and 4 and 5 as positive. Samples of score 3 is ignored. In the dataset, class 1 is the negative and class 2 is the positive. Each class has 1,800,000 training samples and 200,000 testing samples.

The files train.csv and test.csv contain all the training samples as comma-separated values. There are 3 columns in them, corresponding to label/class index (1 or 2), review title and review text. The review title and text are escaped using double quotes ("), and any internal double quote is escaped by 2 double quotes (""). New lines are escaped by a backslash followed with an"n" character, that is "\n".

[25] https://drive.google.com/drive/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M?resourcekey=0-TLwzfR2O-D2aPitmn5o9VQ

1.3.2 Explore Your Data

Building and training a model is only one part of the workflow. Understanding the characteristics of your data beforehand will enable you to build a better model. This could simply mean obtaining a higher accuracy. It could also mean requiring less data for training, or fewer computational resources.

1.3.2.1 Load the Dataset

First up, let's load the dataset into R.

In the dataset, class 1 is the negative and class 2 is the positive review. We will change these to 0 and 1.

columns = (0, 1, 2) # 0 - label/class index, 1 - title/subject, 2 - text body/review.

In this project we will NOT be using the "title" data. We will use only "label" and "text". Also note that I have more comments in the code file/s than in the pdf document.

```
untar("amazon_review_polarity_csv.tar.gz", list = TRUE)  ## check contents
[1] "amazon_review_polarity_csv/"
[2] "amazon_review_polarity_csv/test.csv"
[3] "amazon_review_polarity_csv/train.csv"
[4] "amazon_review_polarity_csv/readme.txt"
untar("amazon_review_polarity_csv.tar.gz")
```

1.3.2.2 Check the Data After loading the data, it's good practice to run some checks on it: pick a few samples and manually check if they are consistent with your expectations. For example see Table 1

```
glimpse(amazon_train)
Rows: 2,879,960
Columns: 3
$ label <dbl> 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0~
$ title <chr> "For Older Mac Operating Systems Only", "greener today than yest~
$ text  <chr> "Does not work on Mac OSX Per Scholastic tech support If you hav~
```

Table 1: Amazon Train data

label	title	text
0	For Older Mac Operating Systems Only	Does not work on Mac OSX Per Scholastic tech support If you have a newer Mac machin
1	greener today than yesterday	I was a skeptic however I nowhave three newspapers the bible six books and two magazir
0	Overrated very overrated	This film is a disappointment Based on the reviews I read here at Amazon and in other r
1	well really	dmst are my favourite band and have been for a while now just mind blowing i wanted t
1	dynomax super turbo exhaust	it fit good took only about mins to put on little quiter than i wanted (but for the price
1	East LA Marine the Guy Gabaldon Story	This movie puts history in perspective for those who know the story of Guy Gabaldon I
0	World of Bad Support	Before getting this game be sure to check out the Support forums WoW (World of warcr
0	disapointing	Only two songs are great Desire All I want is you There are some good live performaces
1	SAITEK X FLIGHT CONTROL SYSTEM BLOWS YOU AWAY	When I purchased my Flight Simulator Deluxe Edition I chose to purchase these controls
1	the secret of science	The best kept secret of science is how strongly it points towards a creator and dovetails

Labels : Negative reviews = 0, Positive reviews = 1

```
unique(amazon_train$label)
[1] 0 1
```

1.3.2.3 Collect Key Metrics Once you've verified the data, collect the following important metrics that can help characterize your text classification problem:

1.Number of samples: Total number of examples you have in the data.

2.Number of classes: Total number of topics or categories in the data.

3.Number of samples per class: Number of samples per class (topic/category). In a balanced dataset, all classes will have a similar number of samples; in an imbalanced dataset, the number of samples in each class will vary widely.

4.Number of words per sample: Median number of words in one sample.

5.Frequency distribution of words: Distribution showing the frequency (number of occurrences) of each word in the dataset.

6.Distribution of sample length: Distribution showing the number of words per sample in the dataset.

Number of samples

```
(num_samples <- nrow(amazon_train))
[1] 2879960
```

Number of classes

```
(num_classes <- length(unique(amazon_train$label)))
[1] 2
```

Number of samples per class

```
# Pretty Balanced classes
(num_samples_per_class <- amazon_train %>%
    count(label))
```

label	n
0	1439405
1	1440555

Number of words per sample

```
amazon_train_text_wordCount <- sapply(temp, length)

(mean_num_words_per_sample <- mean(amazon_train_text_wordCount))
[1] 75.89602

(median_num_words_per_sample <- median(amazon_train_text_wordCount))
[1] 67
```

Table 2: Frequency distribution of words

word	n	total	rank	term frequency
the	11106073	218378699	1	0.0508569
i	6544247	218378699	2	0.0299674
and	6018678	218378699	3	0.0275607
a	5452771	218378699	4	0.0249693
to	5398243	218378699	5	0.0247196
it	5028997	218378699	6	0.0230288
of	4325341	218378699	7	0.0198066
this	4083354	218378699	8	0.0186985
is	3850538	218378699	9	0.0176324
in	2594242	218378699	10	0.0118796

1.3.2.4 Tokenization To build features for supervised machine learning from natural language, we need some way of representing raw text as numbers so we can perform computation on them. Typically, one of the first steps in this transformation from natural language to feature, or any of kind of text analysis, is tokenization. Knowing what tokenization and tokens are, along with the related concept of an n-gram, is important for almost any natural language processing task.

Tokenization in NLP/Text Classification is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

For Frequency distribution of words(nrams) and for Top 25 words see Table 2 and Figure 1

```
Warning: Ignoring unknown parameters: binwidth
```

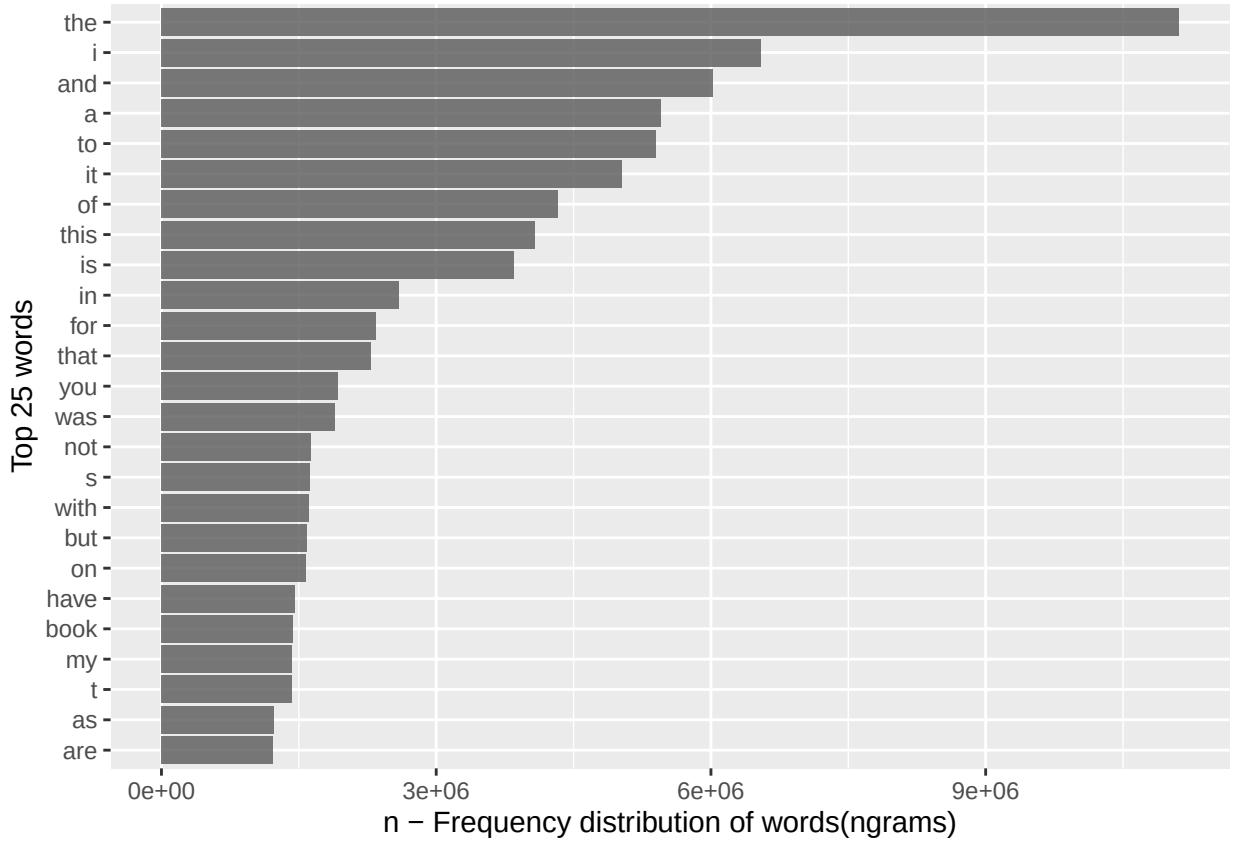

Figure 1: Frequency distribution of words(nrams) for Top 25 words

Table 3: Frequency distribution of words excluding stopwords

word	n	total	rank	term frequency
book	1441251	71253939	1	0.0202270
read	513081	71253939	2	0.0072007
time	506943	71253939	3	0.0071146
movie	431317	71253939	4	0.0060532
love	332012	71253939	5	0.0046596
product	306813	71253939	6	0.0043059
bought	292201	71253939	7	0.0041008
album	265835	71253939	8	0.0037308
story	264836	71253939	9	0.0037168
music	235009	71253939	10	0.0032982

1.3.2.5 Stopwords Once we have split text into tokens, it often becomes clear that not all words carry the same amount of information, if any information at all, for a predictive modeling task. Common words that carry little (or perhaps no) meaningful information are called stop words. It is common advice and practice to remove stop words for various NLP tasks.

The concept of stop words has a long history with Hans Peter Luhn credited with coining the term in 1960. Luhn, H. P. 1960. "Key Word-in-Context Index for Technical Literature (kwic Index)." American Documentation 11 (4): 288–295. doi:10.1002/asi.5090110403[26]. Examples of these words in English are "a," "the," "of," and "didn't." These words are very common and typically don't add much to the meaning of a text but instead ensure the structure of a sentence is sound.

Historically, one of the main reasons for removing stop words was to decrease the computational time for text mining; it can be regarded as a dimensionality reduction of text data and was commonly used in search engines to give better results Huston, Samuel, and W. Bruce Croft. 2010. "Evaluating Verbose Query Processing Techniques." In Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 291–298. SIGIR '10. New York, NY, USA: ACM. doi:10.1145/1835449.1835499[27].

For Frequency distribution of words(ngrams) and for Top 25 words excluding stopwords see Table 3 and Figure 2

Using Pre-made stopwords

```
length(stopwords(source = "smart"))
[1] 571
length(stopwords(source = "snowball"))
[1] 175
length(stopwords(source = "stopwords-iso"))
[1] 1298
```

Frequency distribution of words with stopwords removed

We will use the "stopwords-iso" Pre-made stopwords along with a few unique to our case

```
mystopwords <- c("s", "t", "m", "ve", "re", "d", "ll")
```

[26] https://doi.org/10.1002/asi.5090110403
[27] https://doi.org/10.1145/1835449.1835499

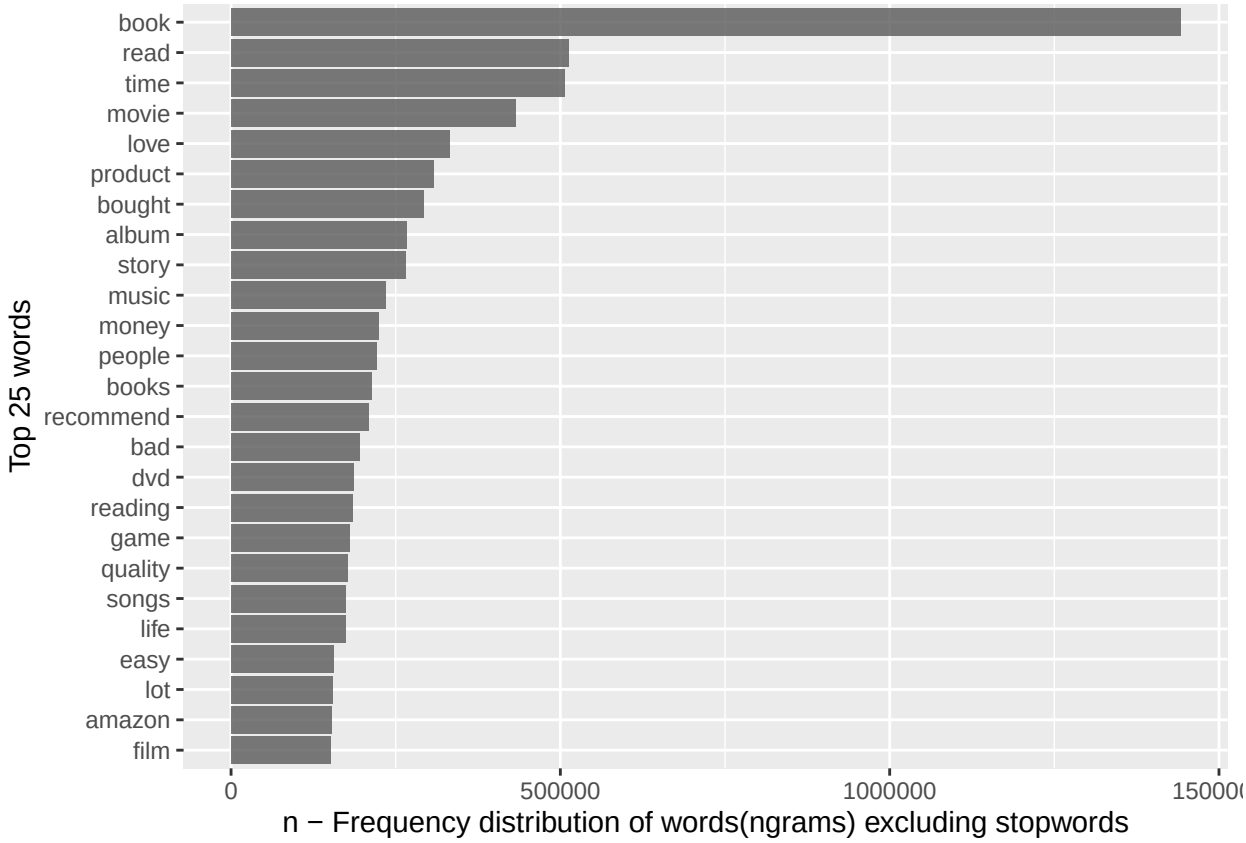

Figure 2: Frequency distribution of words(nrams) for Top 25 words excluding stopwords

Here are Google's recommendations after decades of research:

Algorithm for Data Preparation and Model Building

1. Calculate the number of samples/number of words per sample ratio.
2. If this ratio is less than 1500, tokenize the text as n-grams and use a simple multi-layer perceptron (MLP) model to classify them (left branch in the flowchart below):

a. Split the samples into word n-grams; convert the n-grams into vectors.
b. Score the importance of the vectors and then select the top 20K using the scores.
c. Build an MLP model.

3. If the ratio is greater than 1500, tokenize the text as sequences and use a sepCNN[28] model to classify them (right branch in the flowchart below):

a. Split the samples into words; select the top 20K words based on their frequency.
b. Convert the samples into word sequence vectors.
c. If the original number of samples/number of words per sample ratio is less than 15K, using a fine-tuned pre-trained embedding with the sepCNN model will likely provide the best results.

4. Measure the model performance with different hyperparameter values to find the best model configuration for the dataset.

```
# 3. If the ratio is greater than 1500, tokenize the text as sequences and use
# a sepCNN model see above

(S_W_ratio <- num_samples/median_num_words_per_sample)
[1] 42984.48
```

[28] https://developers.google.com/machine-learning/glossary?utm_source=DevSite&utm_campaign=Text-Class-Guide&utm_medium=referral&utm_content=glossary&utm_term=sepCNN#depthwise-separable-convolutional-neural-network-sepcnn

1.4 Preprocessing for deep learning continued with more exploration

For "Number of words per review text" see Figure 3

For "Number of words per review title" see Figure 4

For "Number of words per review text by label" see Figure 5

For "Number of words per review title by label" see Figure 6

For "Sample/Subset of our training dataset" see Table 4

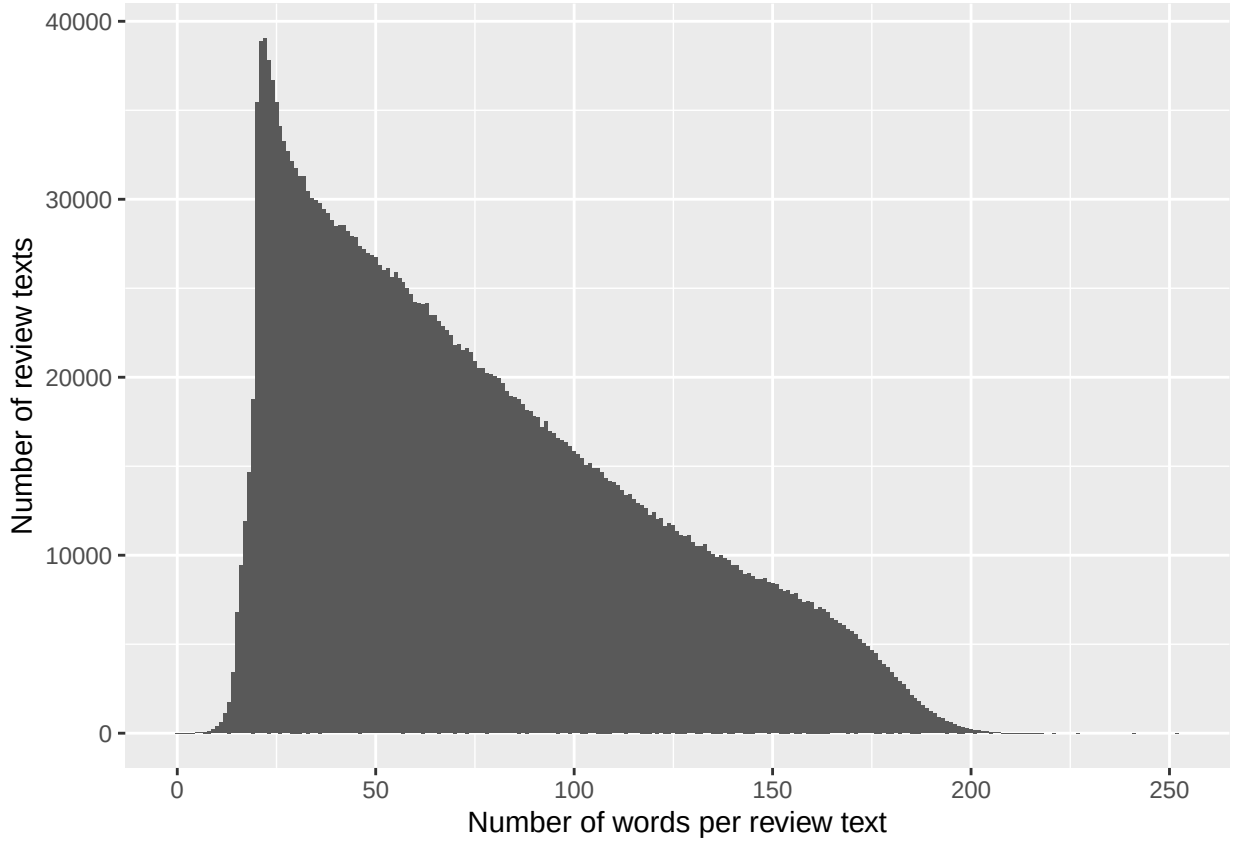

Figure 3: Number of words per review text

Let's trim down our training dataset due to computing resource limitations.

```
amazon_subset_train <- amazon_train %>%
    select(-title) %>%
    mutate(n_words = tokenizers::count_words(text)) %>%
    filter((n_words < 35) & (n_words > 5)) %>%
    select(-n_words)
dim(amazon_subset_train)
[1] 579545      2
# head(amazon_subset_train)
```

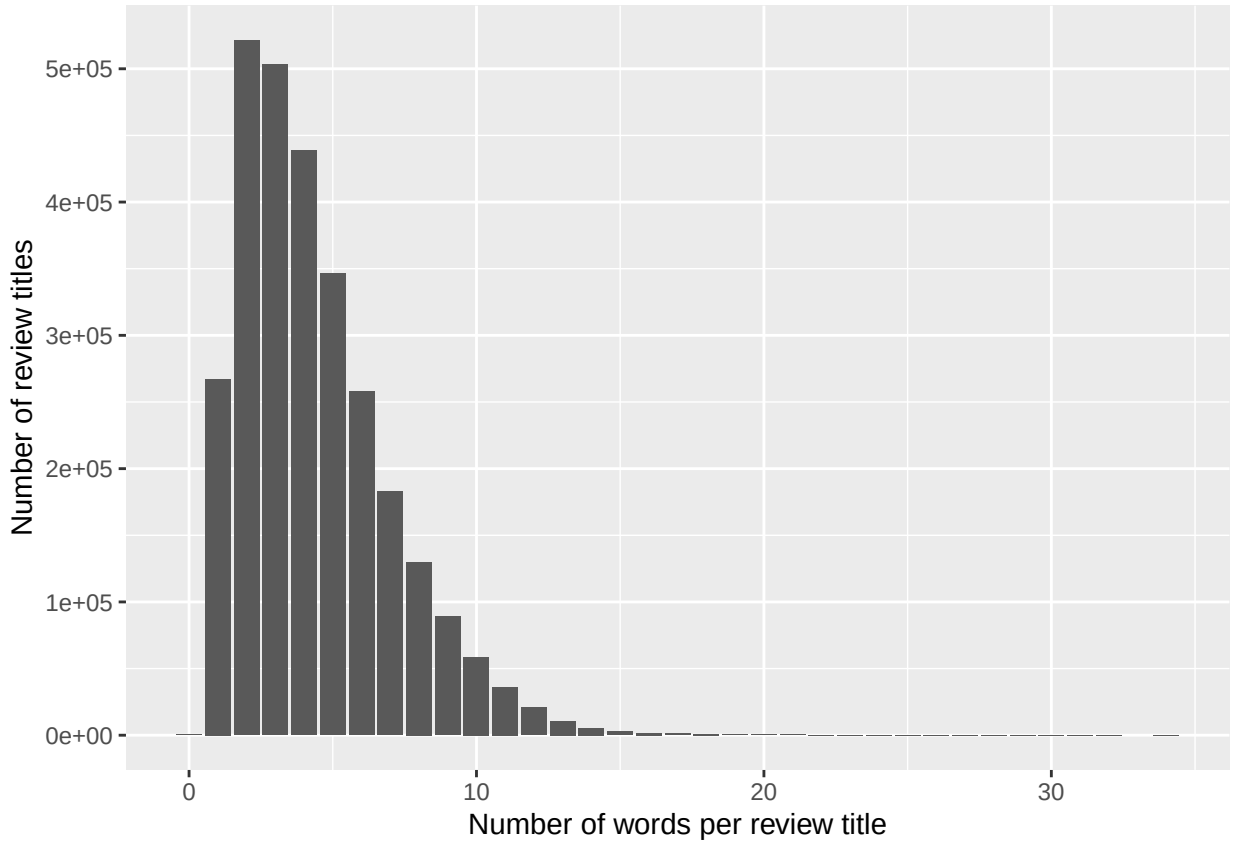

Figure 4: Number of words per review title

Table 4: Sample/Subset of our training dataset

label	text
1	dmst are my favourite band and have been for a while now just mind blowing i wanted to make one correction to t
1	The best kept secret of science is how strongly it points towards a creator and dovetails with Christianity In this m
1	Our children (age) love these DVD s They are somewhat educational and seem to be much better than the cartoo
1	I have enjoyed using these picks I am a beginning guitar player and using a thin gauge pick like this makes strumm
1	This book is very concise and useful I found it very easy to accurately translate quickly
1	Please don t deny your selfof a most irresistable chocolate This biography Am confident to sayForget what s curren
0	These are not clear Not even close They are opaque (and even closer to white) and not advertised as such To me th
1	Comfortable and classy but they scratch really easy Other than that a good buy if you don t plan on wearing them
0	I read about of the book then finally dropped it because I found it rather tiring (I had read Five Children and It b
0	I bought to look up the rules of Euchre and it had hardly anything here I thought the book sucked

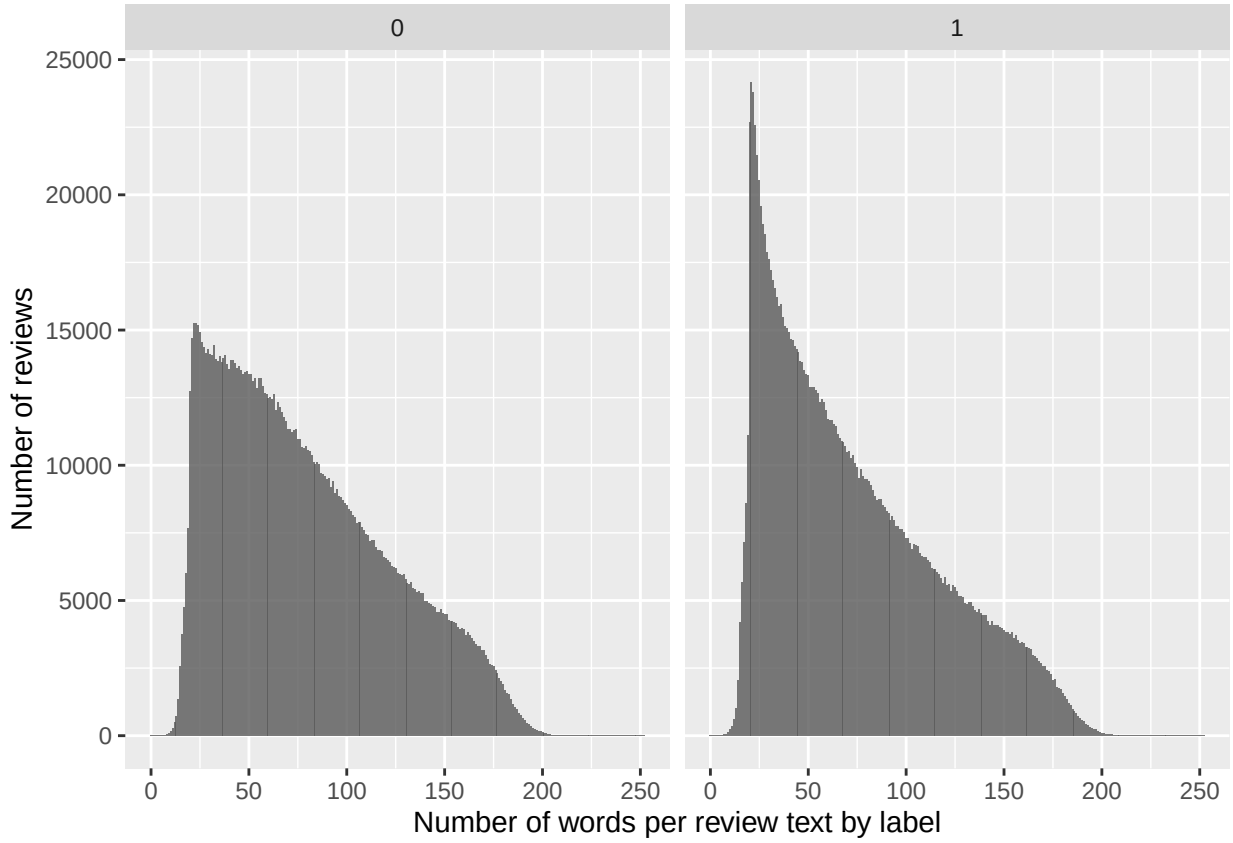

Figure 5: Number of words per review text by label

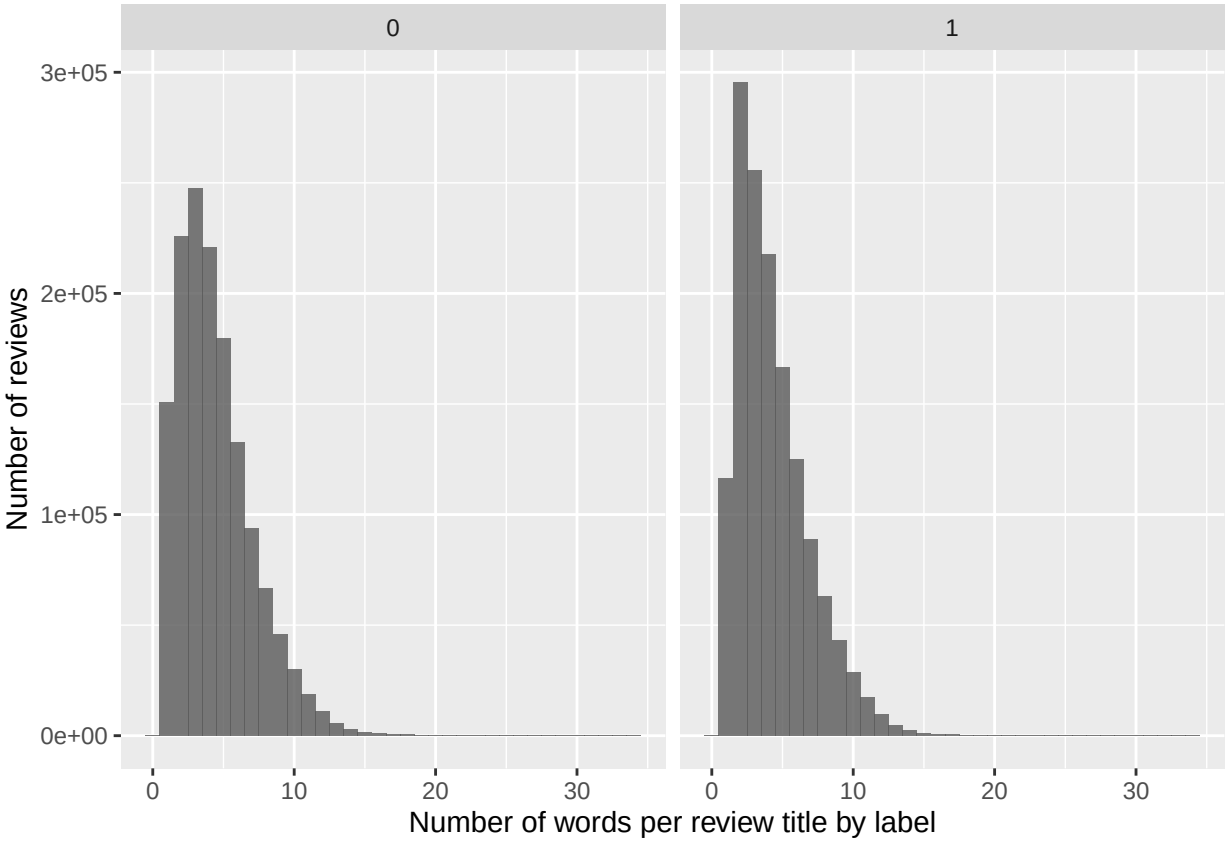

Figure 6: Number of words per review title by label

2 Model Baseline linear classifier

This model serves the purpose of comparison with the deep learning techniques we will implement later on, and also as a succinct summary of a basic supervised machine learning analysis for text.

This linear baseline is a regularized linear model trained on the same data set, using tf-idf weights and 5000 tokens.

2.1 Modify label column to factor

```
# Free computer resources
rm(amazon_train, amazon_val, amazon_train_text_wordCount, num_samples_per_class,
    temp, total_words, train_words)
rm(mean_num_words_per_sample, median_num_words_per_sample, num_classes, num_samples,
    S_W_ratio)
gc()
           used  (Mb) gc trigger   (Mb)   max used    (Mb)
Ncells  3372585 180.2   16004112  854.8   20005140  1068.4
Vcells 24396993 186.2 1240182133 9461.9 1550227666 11827.3

# save(amazon_subset_train)
write_csv(amazon_subset_train, "amazon_review_polarity_csv/amazon_subset_train.csv",
    col_names = TRUE)

amazon_train <- amazon_subset_train

amazon_train <- amazon_train %>%
    mutate(label = as.factor(label))

# amazon_val <- amazon_train %>% mutate(label = as.factor(label))
```

2.2 Split into test/train and create resampling folds

```
set.seed(1234)
amazon_split <- amazon_train %>%
    initial_split()
amazon_train <- training(amazon_split)
amazon_test <- testing(amazon_split)
set.seed(123)
amazon_folds <- vfold_cv(amazon_train)
# amazon_folds
```

2.3 Recipe for data preprocessing

"step_tfidf" creates a specification of a recipe step that will convert a tokenlist into multiple variables containing the term frequency-inverse document frequency[29] of tokens.(check it out in the console by typing ?textrecipes::step_tfidf)

```
# library(textrecipes)
```

[29] https://www.tidytextmining.com/tfidf.html

```
amazon_rec <- recipe(label ~ text, data = amazon_train) %>%
    step_tokenize(text) %>%
    step_tokenfilter(text, max_tokens = 5000) %>%
    step_tfidf(text)

amazon_rec
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor          1

Operations:

Tokenization for text
Text filtering for text
Term frequency-inverse document frequency with text
```

2.4 Lasso regularized classification model and tuning

Linear models are not considered cutting edge in NLP research, but are a workhorse in real-world practice. Here we will use a lasso regularized model Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." Journal of the Royal Statistical Society. Series B (Methodological) 58 (1). Royal Statistical Society, Wiley: 267–288.[30].

Let's create a specification of lasso regularized model.

"penalty" is a model hyperparameter and we cannot learn its best value during model training, but we can estimate the best value by training many models on resampled data sets and exploring how well all these models perform. Let's build a new model specification for model tuning.

```
lasso_spec <- logistic_reg(penalty = tune(), mixture = 1) %>%
    set_mode("classification") %>%
    set_engine("glmnet")

lasso_spec
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = 1

Computational engine: glmnet
```

2.5 A model workflow

We need a few more components before we can tune our workflow. Let's use a sparse data encoding.

We can change how our text data is represented to take advantage of its sparsity, especially for models like lasso regularized models. The regularized regression model we trained above used set_engine("glmnet");

[30] http://www.jstor.org/stable/2346178

this computational engine can be more efficient when text data is transformed to a sparse matrix, rather than a dense data frame or tibble representation.

To keep our text data sparse throughout modeling and use the sparse capabilities of set_engine("glmnet"), we need to explicitly set a non-default preprocessing blueprint, using the package hardhat Vaughan, Davis, and Max Kuhn. 2020. hardhat: Construct Modeling Packages.[31].

```
library(hardhat)
sparse_bp <- default_recipe_blueprint(composition = "dgCMatrix")
```

Let's create a grid of possible regularization penalties to try, using a convenience function for penalty() called grid_regular() from the dials package.

```
lambda_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 20)
lambda_grid
```

penalty
0.0000100
0.0000183
0.0000336
0.0000616
0.0001129
0.0002069
0.0003793
0.0006952
0.0012743
0.0023357
0.0042813
0.0078476
0.0143845
0.0263665
0.0483293
0.0885867
0.1623777
0.2976351
0.5455595
1.0000000

Now these can be combined in a tuneable workflow()

```
amazon_wf <- workflow() %>%
    add_recipe(amazon_rec, blueprint = sparse_bp) %>%
    add_model(lasso_spec)

amazon_wf
== Workflow ==========================================================================
Preprocessor: Recipe
Model: logistic_reg()

-- Preprocessor ------------------------------------------------------------------
3 Recipe Steps
```

[31] https://CRAN.R-project.org/package=hardhat

```
* step_tokenize()
* step_tokenfilter()
* step_tfidf()

-- Model ---------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = 1

Computational engine: glmnet
```

2.6 Tune the workflow

Let's use tune_grid() to fit a model at each of the values for the regularization penalty in our regular grid and every resample in amazon_folds.

```
set.seed(2020)
lasso_rs <- tune_grid(amazon_wf, amazon_folds, grid = lambda_grid, control = control_resamples(save_pred

# lasso_rs
```

We now have a set of metrics for each value of the regularization penalty.

We can extract the relevant information using collect_metrics() and collect_predictions()

See Table 5 for Lasso Metrics

```
m_lm <- collect_metrics(lasso_rs)
kable(m_lm, format = "simple", caption = "Lasso Metrics\\label{tbl:lasso_metrics}")
```

Table 5: Lasso Metrics

penalty	.metric	.estimator	mean	n	std_err	.config
0.0000100	accuracy	binary	0.8893912	10	0.0005032	Preprocessor1_Model01
0.0000100	roc_auc	binary	0.9538016	10	0.0003006	Preprocessor1_Model01
0.0000183	accuracy	binary	0.8893912	10	0.0005032	Preprocessor1_Model02
0.0000183	roc_auc	binary	0.9538016	10	0.0003006	Preprocessor1_Model02
0.0000336	accuracy	binary	0.8893912	10	0.0004969	Preprocessor1_Model03
0.0000336	roc_auc	binary	0.9538247	10	0.0003008	Preprocessor1_Model03
0.0000616	accuracy	binary	0.8895062	10	0.0004782	Preprocessor1_Model04
0.0000616	roc_auc	binary	0.9539107	10	0.0003016	Preprocessor1_Model04
0.0001129	accuracy	binary	0.8896719	10	0.0004941	Preprocessor1_Model05
0.0001129	roc_auc	binary	0.9540329	10	0.0003014	Preprocessor1_Model05
0.0002069	accuracy	binary	0.8897064	10	0.0004424	Preprocessor1_Model06
0.0002069	roc_auc	binary	0.9541505	10	0.0003024	Preprocessor1_Model06
0.0003793	accuracy	binary	0.8894809	10	0.0004733	Preprocessor1_Model07
0.0003793	roc_auc	binary	0.9541045	10	0.0003048	Preprocessor1_Model07
0.0006952	accuracy	binary	0.8883996	10	0.0004462	Preprocessor1_Model08
0.0006952	roc_auc	binary	0.9534221	10	0.0003069	Preprocessor1_Model08
0.0012743	accuracy	binary	0.8853696	10	0.0005228	Preprocessor1_Model09
0.0012743	roc_auc	binary	0.9512859	10	0.0003173	Preprocessor1_Model09
0.0023357	accuracy	binary	0.8782767	10	0.0005656	Preprocessor1_Model10
0.0023357	roc_auc	binary	0.9465029	10	0.0003413	Preprocessor1_Model10
0.0042813	accuracy	binary	0.8654758	10	0.0005853	Preprocessor1_Model11
0.0042813	roc_auc	binary	0.9375896	10	0.0003789	Preprocessor1_Model11
0.0078476	accuracy	binary	0.8447308	10	0.0006268	Preprocessor1_Model12
0.0078476	roc_auc	binary	0.9225346	10	0.0003517	Preprocessor1_Model12
0.0143845	accuracy	binary	0.8114196	10	0.0008523	Preprocessor1_Model13
0.0143845	roc_auc	binary	0.8982350	10	0.0004307	Preprocessor1_Model13
0.0263665	accuracy	binary	0.7658711	10	0.0008243	Preprocessor1_Model14
0.0263665	roc_auc	binary	0.8620465	10	0.0005772	Preprocessor1_Model14
0.0483293	accuracy	binary	0.7075908	10	0.0008798	Preprocessor1_Model15
0.0483293	roc_auc	binary	0.8020162	10	0.0008407	Preprocessor1_Model15
0.0885867	accuracy	binary	0.6665424	10	0.0010263	Preprocessor1_Model16

penalty	.metric	.estimator	mean	n	std_err	.config
0.0885867	roc_auc	binary	0.7273321	10	0.0006962	Preprocessor1_Model16
0.1623777	accuracy	binary	0.5796028	10	0.0007071	Preprocessor1_Model17
0.1623777	roc_auc	binary	0.5000000	10	0.0000000	Preprocessor1_Model17
0.2976351	accuracy	binary	0.5796028	10	0.0007071	Preprocessor1_Model18
0.2976351	roc_auc	binary	0.5000000	10	0.0000000	Preprocessor1_Model18
0.5455595	accuracy	binary	0.5796028	10	0.0007071	Preprocessor1_Model19
0.5455595	roc_auc	binary	0.5000000	10	0.0000000	Preprocessor1_Model19
1.0000000	accuracy	binary	0.5796028	10	0.0007071	Preprocessor1_Model20
1.0000000	roc_auc	binary	0.5000000	10	0.0000000	Preprocessor1_Model20

What are the best models?

See Table 6 for Best Lasso ROC.

```
m_blr <- show_best(lasso_rs, "roc_auc")
kable(m_blr, format = "simple", caption = "Best Lasso ROC\\label{tbl:best_lasso_roc}")
```

Table 6: Best Lasso ROC

penalty	.metric	.estimator	mean	n	std_err	.config
0.0002069	roc_auc	binary	0.9541505	10	0.0003024	Preprocessor1_Model06
0.0003793	roc_auc	binary	0.9541045	10	0.0003048	Preprocessor1_Model07
0.0001129	roc_auc	binary	0.9540329	10	0.0003014	Preprocessor1_Model05
0.0000616	roc_auc	binary	0.9539107	10	0.0003016	Preprocessor1_Model04
0.0000336	roc_auc	binary	0.9538247	10	0.0003008	Preprocessor1_Model03

See Table 7 for Best Lasso Accuracy.

```
m_bla <- show_best(lasso_rs, "accuracy")
kable(m_bla, format = "simple", caption = "Best Lasso Accuracy\\label{tbl:best_lasso_acc}")
```

Table 7: Best Lasso Accuracy

penalty	.metric	.estimator	mean	n	std_err	.config
0.0002069	accuracy	binary	0.8897064	10	0.0004424	Preprocessor1_Model06
0.0001129	accuracy	binary	0.8896719	10	0.0004941	Preprocessor1_Model05
0.0000616	accuracy	binary	0.8895062	10	0.0004782	Preprocessor1_Model04
0.0003793	accuracy	binary	0.8894809	10	0.0004733	Preprocessor1_Model07
0.0000100	accuracy	binary	0.8893912	10	0.0005032	Preprocessor1_Model01

Let's visualize these metrics; accuracy and ROC AUC, in Figure 7 to see what the best model is.

See Table 8 for Lasso Predictions

```
m_lp <- collect_predictions(lasso_rs)
kable(head(m_lp), format = "simple", caption = "Lasso Predictions\\label{tbl:lasso_predictions}")
```

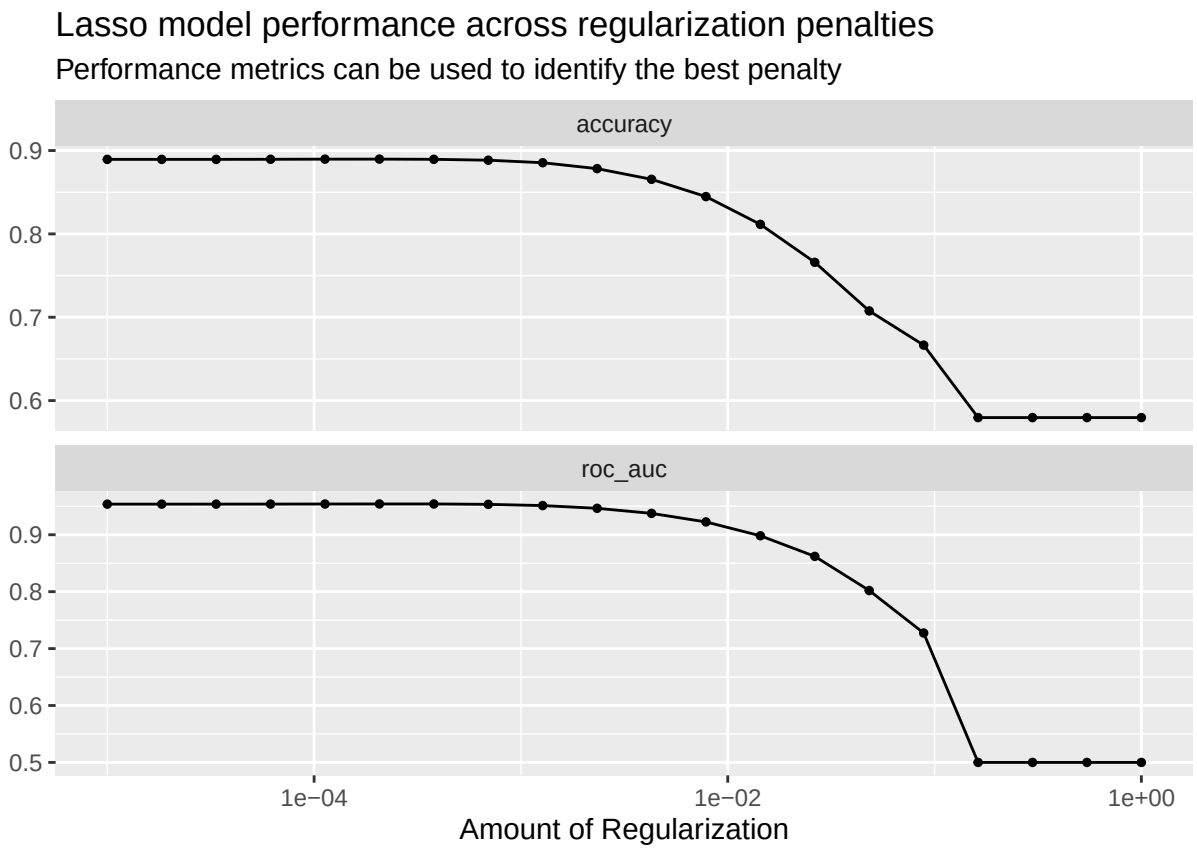

Figure 7: Lasso model performance across regularization penalties

Table 8: Lasso Predictions

id	.pred_0	.pred_1	.row	penalty	.pred_class	label	.config
Fold01	0.4909890	0.5090110	10	1e-05	1	0	Preprocessor1_Model01
Fold01	0.0116122	0.9883878	28	1e-05	1	1	Preprocessor1_Model01
Fold01	0.0145242	0.9854758	30	1e-05	1	1	Preprocessor1_Model01
Fold01	0.0144831	0.9855169	68	1e-05	1	1	Preprocessor1_Model01
Fold01	0.2691942	0.7308058	79	1e-05	1	0	Preprocessor1_Model01
Fold01	0.9884782	0.0115218	86	1e-05	0	0	Preprocessor1_Model01

Figure 8 shows the ROC curve, a visualization of how well a classification model can distinguish between classes

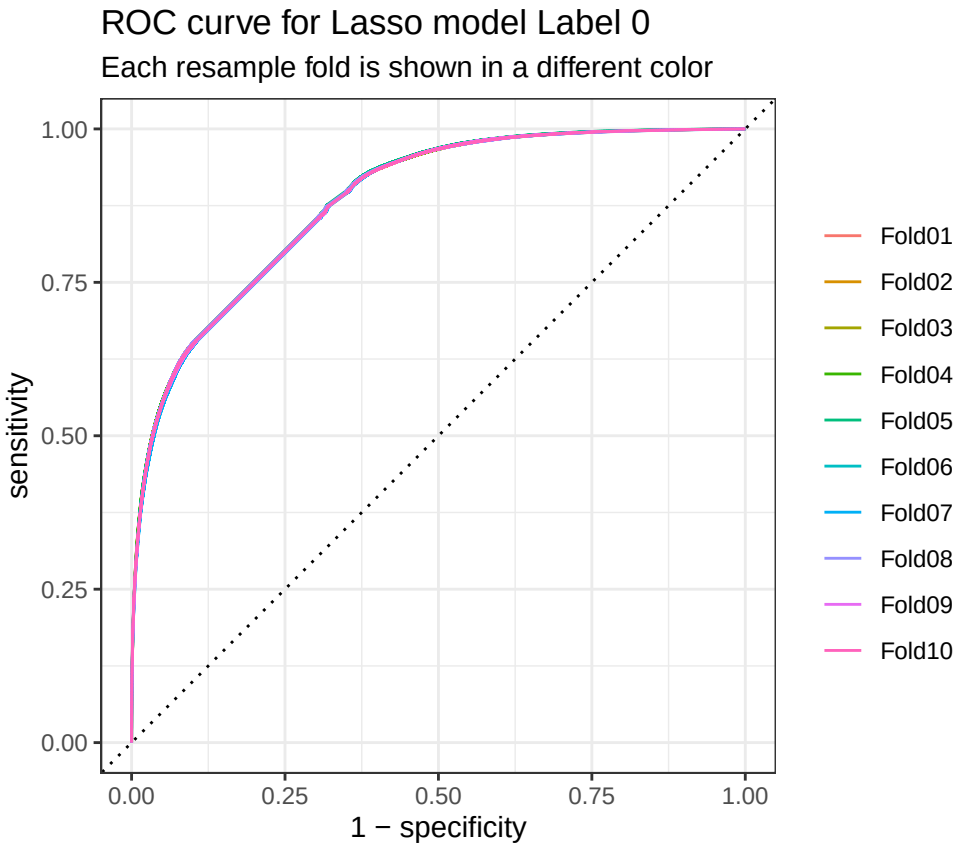

Figure 8: Lasso model ROC Label 0

Figure 9 shows the ROC curve, a visualization of how well a classification model can distinguish between classes

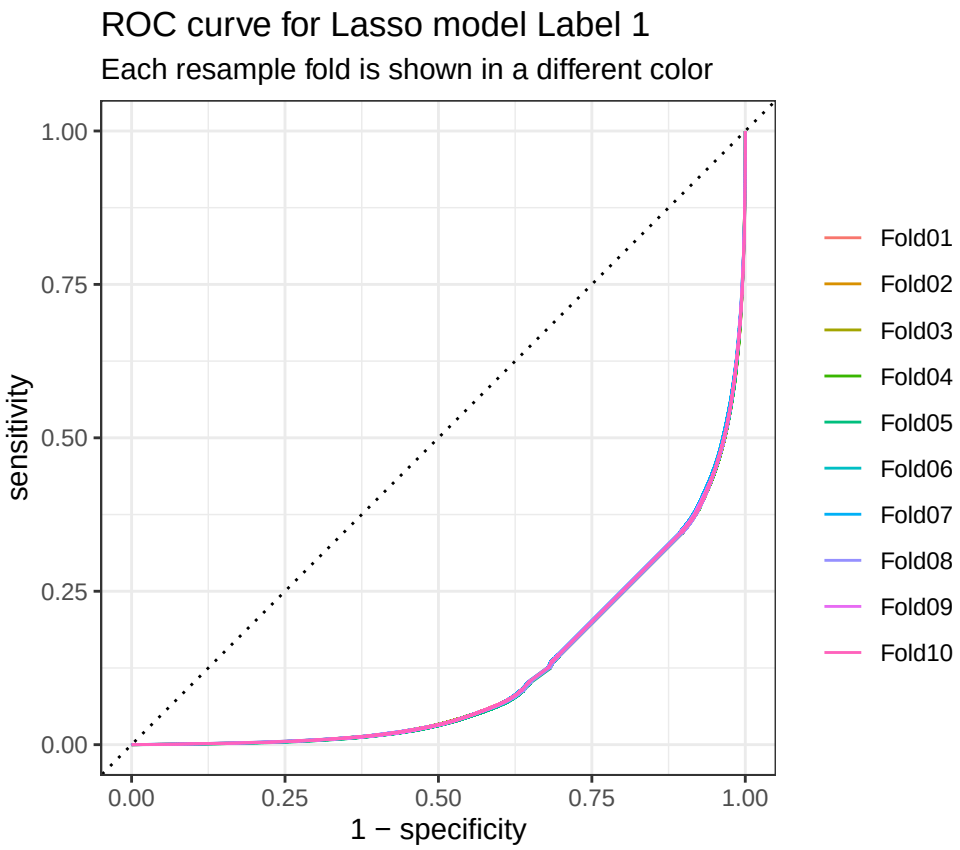

Figure 9: Lasso model ROC Label 1

2.7 Results

We saw that regularized linear models, such as lasso, often work well for text data sets.

The default performance parameters for binary classification are accuracy and ROC AUC (area under the receiver operator characteristic curve). Here, the best accuracy is:

Best ROC_AUC is 0.9541505
Best Accuracy is 0.8897064

As we go along, we will be comparing different approaches. Let's start by creating a results table with this BLM to get Table 9:

Table 9: Baseline Linear Model Results

Index	Method	Accuracy	Loss
1	BLM	0.8897064	NA

Accuracy and ROC AUC are performance metrics used for classification models. For both, values closer to 1 are better.

Accuracy is the proportion of the data that are predicted correctly. Be aware that accuracy can be misleading in some situations, such as for imbalanced data sets.

ROC AUC measures how well a classifier performs at different thresholds. The ROC curve plots the true positive rate against the false positive rate, and AUC closer to 1 indicates a better-performing model while AUC closer to 0.5 indicates a model that does no better than random guessing.

Figure 8 and Figure 9 show the ROC curves, a visualization of how well our classification model can distinguish between classes.

The area under each of these curves is the roc_auc metric we have computed. If the curve was close to the diagonal line, then the model's predictions would be no better than random guessing.

One metric alone cannot give you a complete picture of how well your classification model is performing. The confusion matrix is a good starting point to get an overview of your model performance, as it includes rich information.

Another way to evaluate our model is to evaluate the confusion matrix. A confusion matrix tabulates a model's false positives and false negatives for each class. The function conf_mat_resampled() computes a separate confusion matrix for each resample and takes the average of the cell counts. This allows us to visualize an overall confusion matrix rather than needing to examine each resample individually.

3 Preprocessing for rest of the models

Preprocessing for deep learning models is different than preprocessing for most other text models. These neural networks model sequences, so we have to choose the length of sequences we would like to include. Sequences that are longer than this length are truncated (information is thrown away) and those that are shorter than this length are padded with zeroes (an empty, non-informative value) to get to the chosen sequence length. This sequence length is a hyperparameter of the model and we need to select this value such that we don't overshoot and introduce a lot of padded zeroes which would make the model hard to train, or undershoot and cut off too much informative text.

We will use the recipes and textrecipes packages for data preprocessing and feature engineering.

The formula used to specify this recipe ~ text does not have an outcome, because we are using recipes and textrecipes functions on their own, outside of the rest of the tidymodels framework; we don't need to

know about the outcome here. This preprocessing recipe tokenizes our text and filters to keep only the top 20,000 words and then it transforms the tokenized text into a numeric format appropriate for modeling, using step_sequence_onehot().

```
rm(amazon_folds, amazon_rec, amazon_split, amazon_test, amazon_train, amazon_wf,
    lambda_grid, lasso_rs, lasso_spec, sparse_bp)

gc()
            used  (Mb) gc trigger   (Mb)   max used    (Mb)
Ncells   4147050 221.5   12861945  687.0   20005140  1068.4
Vcells 112719032 860.0  867481767 6618.4 4028113514 30732.1

amazon_subset_train <- readr::read_csv("amazon_review_polarity_csv/amazon_subset_train.csv")

amazon_train <- amazon_subset_train

max_words <- 20000
max_length <- 30
mystopwords <- c("s", "t", "m", "ve", "re", "d", "ll")

amazon_rec <- recipe(~text, data = amazon_subset_train) %>%
    step_text_normalization(text) %>%
    step_tokenize(text) %>%
    step_stopwords(text, stopword_source = "stopwords-iso", custom_stopword_source = mystopwords) %>%
    step_tokenfilter(text, max_tokens = max_words) %>%
    step_sequence_onehot(text, sequence_length = max_length)

amazon_rec
Data Recipe

Inputs:

      role #variables
 predictor          1

Operations:

text_normalizationming for text
Tokenization for text
Stop word removal for text
Text filtering for text
Sequence 1 hot encoding for text
```

The prep() function will compute or estimate statistics from the training set; the output of prep() is a prepped recipe.

When we bake() a prepped recipe, we apply the preprocessing to the data set. We can get out the training set that we started with by specifying new_data = NULL or apply it to another set via new_data = my_other_data_set. The output of bake() is a data set like a tibble or a matrix, depending on the composition argument.

Let's now prepare and apply our feature engineering recipe amazon_rec so we can use it in our deep learning model.

```
amazon_prep <- prep(amazon_rec)
```

```
amazon_subset_train <- bake(amazon_prep, new_data = NULL, composition = "matrix")
dim(amazon_subset_train)
[1] 579545     30
```

The prep() function will compute or estimate statistics from the training set; the output of prep() is a prepped recipe. The prepped recipe can be tidied using tidy() to extract the vocabulary, represented in the vocabulary and token columns.

```
amazon_prep %>%
    tidy(5) %>%
    head(10)
```

terms	vocabulary	token	id
text	1	a	sequence_onehot_bCTRZ
text	2	à	sequence_onehot_bCTRZ
text	3	aa	sequence_onehot_bCTRZ
text	4	aaa	sequence_onehot_bCTRZ
text	5	aaaa	sequence_onehot_bCTRZ
text	6	aaliyah	sequence_onehot_bCTRZ
text	7	aaron	sequence_onehot_bCTRZ
text	8	ab	sequence_onehot_bCTRZ
text	9	abandon	sequence_onehot_bCTRZ
text	10	abandoned	sequence_onehot_bCTRZ

4 Model DNN

A densely connected neural network is one of the simplest configurations for a deep learning model and is typically not a model that will achieve the highest performance on text data, but it is a good place to start to understand the process of building and evaluating deep learning models for text.

In a densely connected neural network, layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives an input from all the neurons present in the previous layer - thus, they're densely connected.

The input comes in to the network all at once and is densely (in this case, fully) connected to the first hidden layer. A layer is "hidden" in the sense that it doesn't connect to the outside world; the input and output layers take care of this. The neurons in any given layer are only connected to the next layer. The numbers of layers and nodes within each layer are variable and are hyperparameters of the model selected by us.

4.1 A Simple flattened dense neural network

Our first deep learning model embeds the Amazon Reviews in sequences of vectors, flattens them, and then trains a dense network layer to predict whether the Review was positive(1) or not(0).

1. We initiate the Keras model as a linear stack of layers with keras_model_sequential().
2. Our first layer - layer_embedding() turns each observation into an (embedding_dim * sequence_length) = 12 * 30
3. In total, we will create a (number_of_observations * embedding_dim * sequence_length) data cube.
4. The next layer_flatten() layer takes the matrix for each observation and flattens them down into one dimension. This will create a (30 * 12) = 360 long vector for each observation.
5. layer_layer_normalization() - Normalize the activations of the previous layer for each given example in a batch independently.
6. Lastly, we have 2 densely connected layers. The last layer has a sigmoid activation function to give us an output between 0 and 1, since we want to model a probability for a binary classification problem.

```
# library(keras) use_python(python =
# '/c/Users/bijoor/.conda/envs/tensorflow-python/python.exe', required = TRUE)
# use_condaenv(condaenv = 'tensorflow-python', required = TRUE)

dense_model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_words + 1, output_dim = 12, input_length = max_length) %>%
    layer_flatten() %>%
    layer_layer_normalization() %>%
    # layer_dropout(0.1) %>%
layer_dense(units = 64) %>%
    # layer_activation_leaky_relu() %>%
layer_activation_relu() %>%
    layer_dense(units = 1, activation = "sigmoid")

dense_model
Model
Model: "sequential"
_________________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
==========================================================================================
embedding (Embedding)                (None, 30, 12)                240012
------------------------------------------------------------------------------------------
flatten (Flatten)                    (None, 360)                   0
------------------------------------------------------------------------------------------
layer_normalization (LayerNormaliza (None, 360)                   720
------------------------------------------------------------------------------------------
dense_1 (Dense)                      (None, 64)                    23104
------------------------------------------------------------------------------------------
re_lu (ReLU)                         (None, 64)                    0
------------------------------------------------------------------------------------------
dense (Dense)                        (None, 1)                     65
==========================================================================================
Total params: 263,901
Trainable params: 263,901
Non-trainable params: 0
------------------------------------------------------------------------------------------
```

Before we can fit this model to the data it requires an optimizer and a loss function to be able to compile.

When the neural network finishes passing a batch of data through the network, it needs a way to use the difference between the predicted values and true values to update the network's weights. The algorithm that determines those weights is known as the optimization algorithm. Many optimizers are available within Keras.

We will choose one of the following based on our previous experimentation.

optimizer_adam() - Adam - A Method for Stochastic Optimization

optimizer_sgd() - Stochastic gradient descent optimizer

We can also use various options during training using the compile() function, such as optimizer, loss and metrics.

```
# opt <- optimizer_adam(lr = 0.0001, decay = 1e-6) opt <- optimizer_sgd(lr =
# 0.001, decay = 1e-6)
opt <- optimizer_sgd()
dense_model %>%
    compile(optimizer = opt, loss = "binary_crossentropy", metrics = c("accuracy"))
```

Finally, we can fit this model.

Here we specify the Keras defaults for creating a validation split and tracking metrics with an internal validation split of 20%.

```
dense_history <- dense_model %>%
    fit(x = amazon_subset_train, y = amazon_train$label, batch_size = 1024, epochs = 50,
        initial_epoch = 0, validation_split = 0.2, verbose = 2)

dense_history
```

```
Final epoch (plot to see history):
        loss: 0.2457
    accuracy: 0.8994
    val_loss: 0.2688
val_accuracy: 0.8898
```

"DNN Model 1 Fit History using validation_split" Figure 10

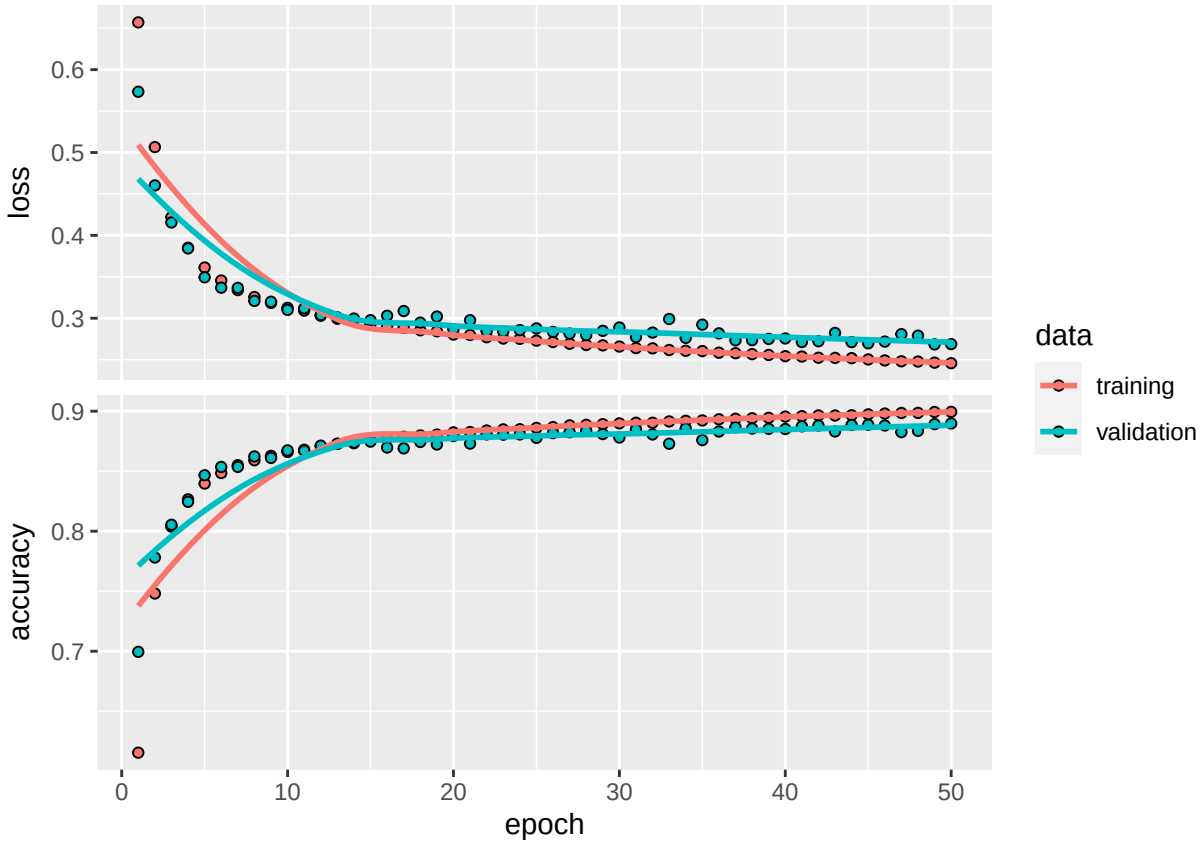

Figure 10: DNN Model 1 Fit History using validation_split

4.2 Evaluation

Instead of using Keras defaults, we can use tidymodels functions to be more specific about these model characteristics. Instead of using the validation_split argument to fit(), we can create our own validation set using tidymodels and use validation_data argument for fit(). We create our validation split from the training set.

```
set.seed(234)
amazon_val_eval <- validation_split(amazon_train, strata = label)
# amazon_val_eval << I am getting a pandoc stack error printing this
```

The split object contains the information necessary to extract the data we will use for training/analysis and the data we will use for validation/assessment. We can extract these data sets in their raw, unprocessed form from the split using the helper functions analysis() and assessment(). Then, we can apply our prepped preprocessing recipe amazon_prep to both to transform this data to the appropriate format for our neural network architecture.

```
amazon_analysis <- bake(amazon_prep, new_data = analysis(amazon_val_eval$splits[[1]]),
    composition = "matrix")
dim(amazon_analysis)
[1] 434658     30
```

```
amazon_assess <- bake(amazon_prep, new_data = assessment(amazon_val_eval$splits[[1]]),
    composition = "matrix")
dim(amazon_assess)
[1] 144887     30
```

Here we get outcome variables for both sets.

```
label_analysis <- analysis(amazon_val_eval$splits[[1]]) %>%
    pull(label)
label_assess <- assessment(amazon_val_eval$splits[[1]]) %>%
    pull(label)
```

Let's setup a new DNN model 2.

Here we use layer_dropout() - Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

```
dense_model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_words + 1, output_dim = 12, input_length = max_length) %>%
    layer_flatten() %>%
    layer_layer_normalization() %>%
    layer_dropout(0.5) %>%
    layer_dense(units = 64) %>%
    layer_activation_relu() %>%
    layer_dropout(0.5) %>%
    layer_dense(units = 128) %>%
    layer_activation_relu() %>%
    layer_dense(units = 128) %>%
    layer_activation_relu() %>%
    layer_dense(units = 1, activation = "sigmoid")
```

```
opt <- optimizer_adam(lr = 1e-04, decay = 1e-06)
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6) opt <- optimizer_sgd()
dense_model %>%
    compile(optimizer = opt, loss = "binary_crossentropy", metrics = c("accuracy"))
```

We now fit this model to validation_data - amazon_assess and label_assess instead of the Keras default validation_split.

```
val_history <- dense_model %>%
    fit(x = amazon_analysis, y = label_analysis, batch_size = 2048, epochs = 20,
        validation_data = list(amazon_assess, label_assess), verbose = 2)

val_history
```

```
Final epoch (plot to see history):
        loss: 0.2638
    accuracy: 0.8932
    val_loss: 0.2837
val_accuracy: 0.8961
```

"DNN Model 2 Fit History using validation_data" Figure 11

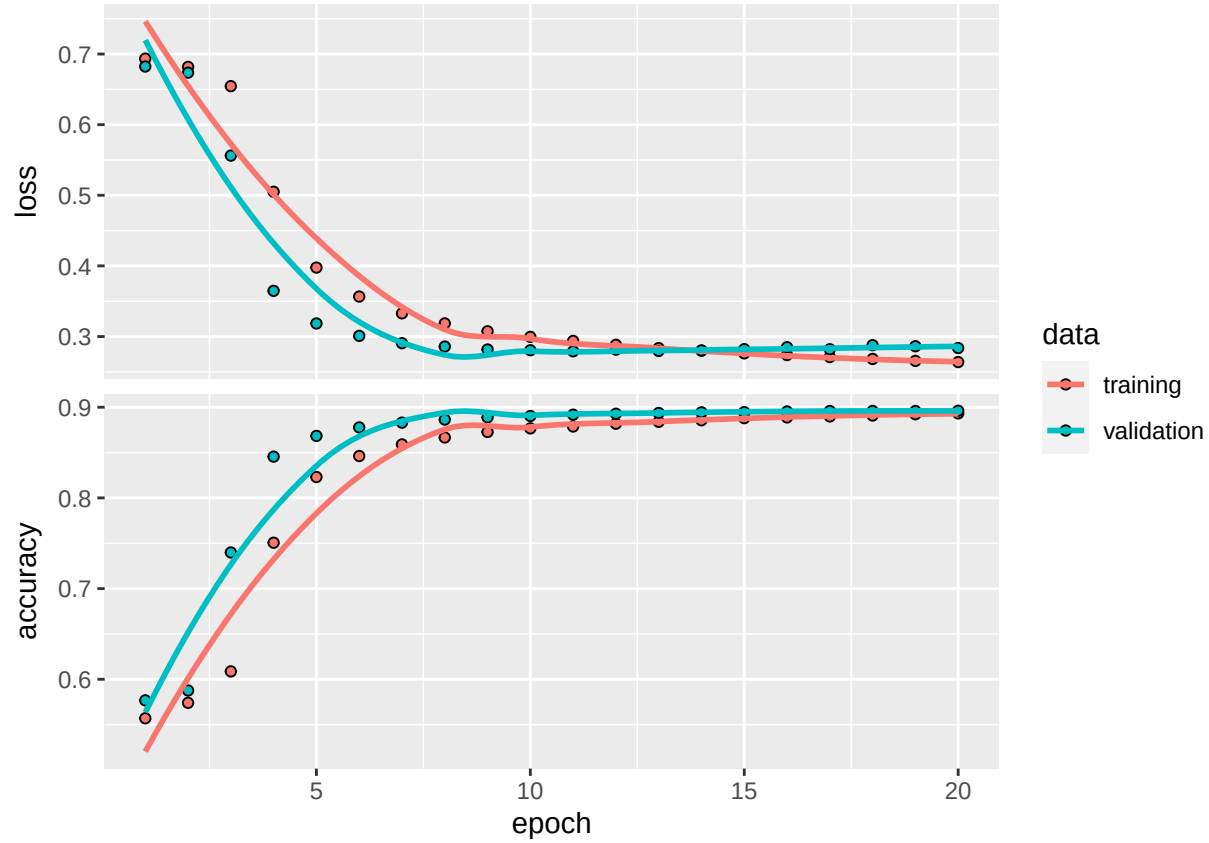

Figure 11: DNN Model 2 Fit History using validation_data

Using our own validation set also allows us to flexibly measure performance using tidymodels functions.

The following function keras_predict() creates prediction results using the Keras model and preprocessed/baked data from using tidymodels, using a 50% probability threshold and works for our binary problem.

```
keras_predict <- function(model, baked_data, response) {
    predictions <- predict(model, baked_data)[, 1]
    tibble(.pred_1 = predictions, .pred_class = if_else(.pred_1 < 0.5, 0, 1), label = response) %>%
        mutate(across(c(label, .pred_class), ~factor(.x, levels = c(1, 0))))
}
```

See Table 10 for "DNN Model 2 Predictions using validation_data"

```
val_res <- keras_predict(dense_model, amazon_assess, label_assess)
# head(val_res)
kable(head(val_res), format = "simple", caption = "DNN Model 2 Predictions using validation data\\label
```

Table 10: DNN Model 2 Predictions using validation data

.pred_1	.pred_class	label
0.0070337	0	0
0.8914716	1	1
0.9580745	1	1
0.3149569	0	0
0.7441973	1	1
0.9910595	1	1

See Table 11 for "DNN Model 2 Metrics using Validation data"

```
m1 <- metrics(val_res, label, .pred_class)
kable(m1, format = "simple", caption = "DNN Model 2 Metrics using Validation data\\label{tbl:val_res_me
```

Table 11: DNN Model 2 Metrics using Validation data

.metric	.estimator	.estimate
accuracy	binary	0.8960983
kap	binary	0.7870405

"DNN Model 2 Confusion Matrix using Validation data" Figure 12

"DNN Model 2 ROC curve using Validation data" Figure 13

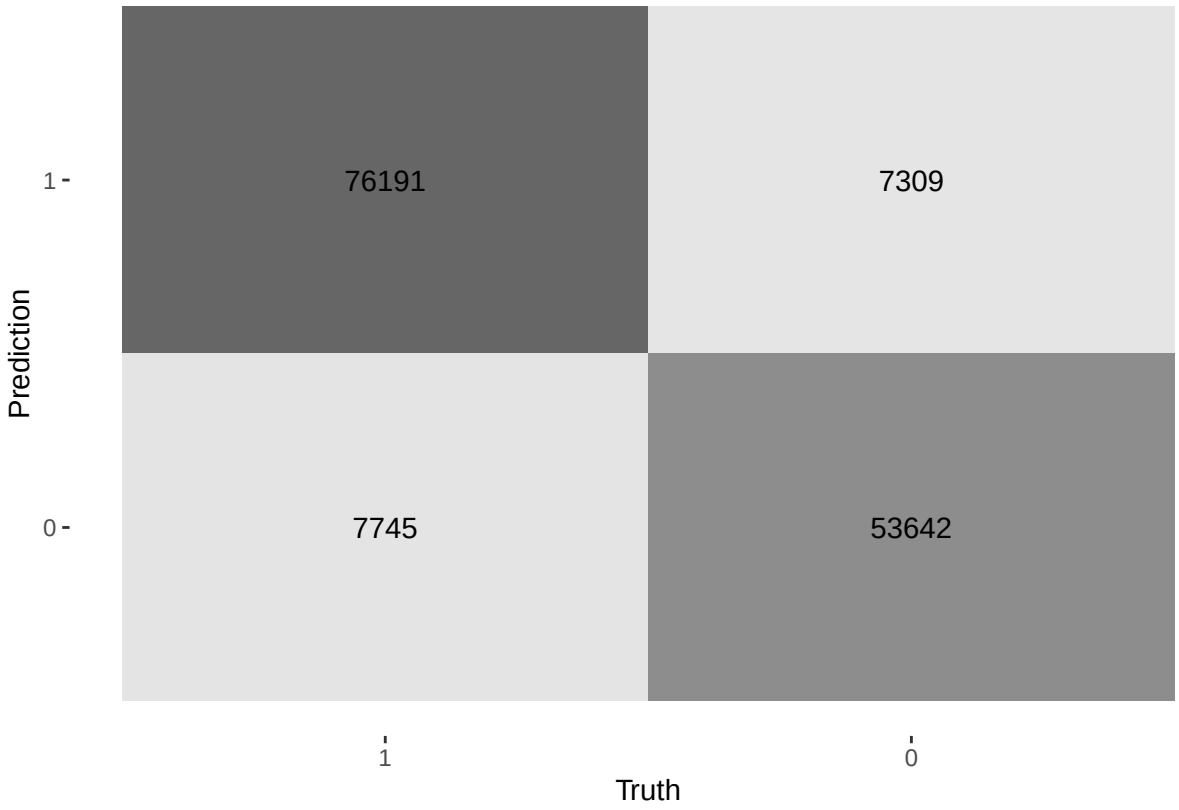

Figure 12: DNN Model 2 Confusion Matrix using Validation data

4.3 Results

DNN model results Table 12:

Table 12: DNN Model Results

Index	Method	Accuracy	Loss
1	BLM	0.8897064	NA
2	DNN	0.8960983	0.2790776

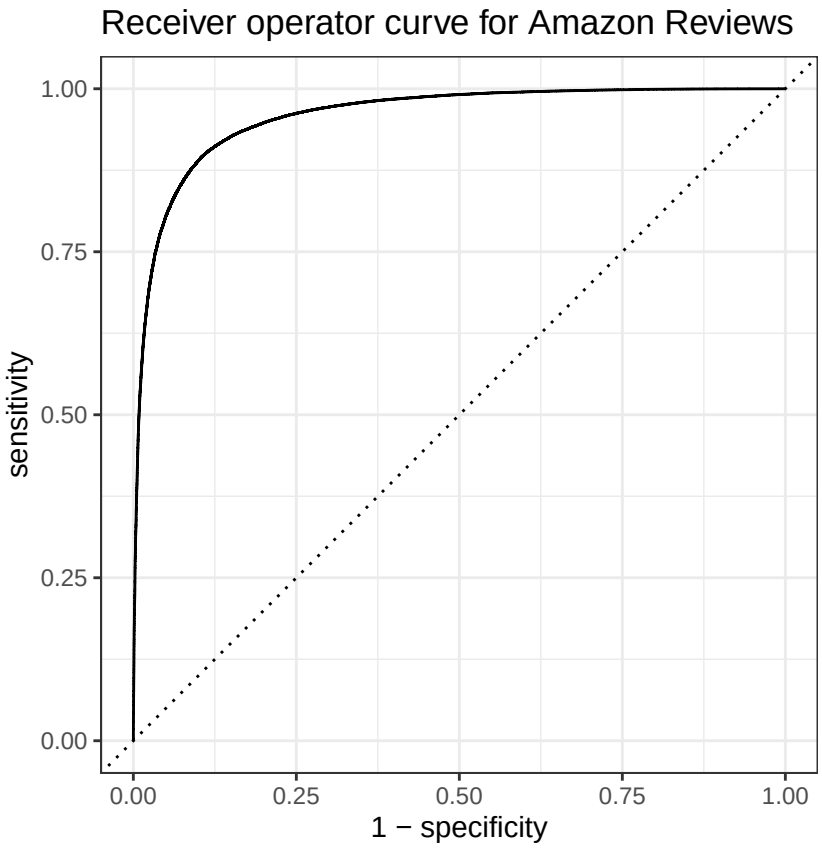

Figure 13: DNN Model 2 ROC curve using Validation data

5 Model CNN

A CNN is a neural network in which at least one layer is a convolutional layer. A typical convolutional neural network consists of some combination of the following layers:

1.Convolutional layers - A layer of a deep neural network in which a convolutional filter passes along an input matrix.
A convolutional operation involves a convolutional filter which is a matrix having the same rank as the input matrix, but a smaller shape and a slice of an input matrix. For example, given a 28x28 input matrix, the filter could be any 2D matrix smaller than 28x28.

For example, in photographic manipulation, all the cells in a convolutional filter are typically set to a constant pattern of ones and zeroes. In machine learning, convolutional filters are typically seeded with random numbers and then the network trains the ideal values.

2.Pooling layers - Reducing a matrix (or matrices) created by an earlier convolutional layer to a smaller matrix. Pooling usually involves taking either the maximum or average value across the pooled area. For example, suppose we have a 3x3 matrix. A pooling operation, just like a convolutional operation, divides that matrix into slices and then slides that convolutional operation by strides. For example, suppose the pooling operation divides the convolutional matrix into 2x2 slices with a 1x1 stride, then four pooling operations take place. Imagine that each pooling operation picks the maximum value of the four in that slice.

Pooling helps enforce translational invariance in the input matrix.

Pooling for vision applications is known more formally as spatial pooling. Time-series applications usually refer to pooling as temporal pooling. Less formally, pooling is often called subsampling or downsampling.

3.Dense layers - just a fully connected layer.

Convolutional neural networks have had great success in certain kinds of problems, especially in image recognition.

5.1 A first CNN model

```
simple_cnn_model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_words + 1, output_dim = 16, input_length = max_length) %>%
    layer_batch_normalization() %>%
    layer_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
    layer_max_pooling_1d(pool_size = 2) %>%
    layer_conv_1d(filter = 64, kernel_size = 3, activation = "relu") %>%
    layer_global_max_pooling_1d() %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1, activation = "sigmoid")

simple_cnn_model
Model
Model: "sequential_2"
_________________________________________________________________________________
Layer (type)                    Output Shape                  Param #
=================================================================================
embedding_2 (Embedding)         (None, 30, 16)                320016
_________________________________________________________________________________
batch_normalization (BatchNormaliza (None, 30, 16)            64
_________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 26, 32)                2592
_________________________________________________________________________________
```

```
max_pooling1d (MaxPooling1D)        (None, 13, 32)                0
_________________________________________________________________________________
conv1d (Conv1D)                     (None, 11, 64)                6208
_________________________________________________________________________________
global_max_pooling1d (GlobalMaxPool (None, 64)                    0
_________________________________________________________________________________
dense_7 (Dense)                     (None, 64)                    4160
_________________________________________________________________________________
dense_6 (Dense)                     (None, 1)                     65
=================================================================================
Total params: 333,105
Trainable params: 333,073
Non-trainable params: 32
_________________________________________________________________________________
```

```
simple_cnn_model %>%
    compile(optimizer = opt, loss = "binary_crossentropy", metrics = c("accuracy"))

simple_cnn_val_history <- simple_cnn_model %>%
    fit(x = amazon_analysis, y = label_analysis, batch_size = 1024, epochs = 7, initial_epoch = 0,
        validation_data = list(amazon_assess, label_assess), verbose = 2)

simple_cnn_val_history
```

```
Final epoch (plot to see history):
        loss: 0.2097
    accuracy: 0.9192
    val_loss: 0.2526
val_accuracy: 0.8983
```

"CNN Model Fit History using validation_data" Figure 14

See Table 13 for "CNN Model Predictions using validation data"

```
simple_cnn_val_res <- keras_predict(simple_cnn_model, amazon_assess, label_assess)
# head(simple_cnn_val_res)
kable(head(simple_cnn_val_res), format = "simple", caption = "CNN Model Predictions using validation da
```

Table 13: CNN Model Predictions using validation data

.pred_1	.pred_class	label
0.0036336	0	0
0.9667417	1	1
0.9940560	1	1
0.1679935	0	0
0.8921552	1	1
0.9995990	1	1

See Table 14 for "CNN Model Metrics using validation data"

```
m2 <- metrics(simple_cnn_val_res, label, .pred_class)
kable(m2, format = "simple", caption = "CNN Model Metrics using validation data\\label{tbl:simple_cnn_va
```

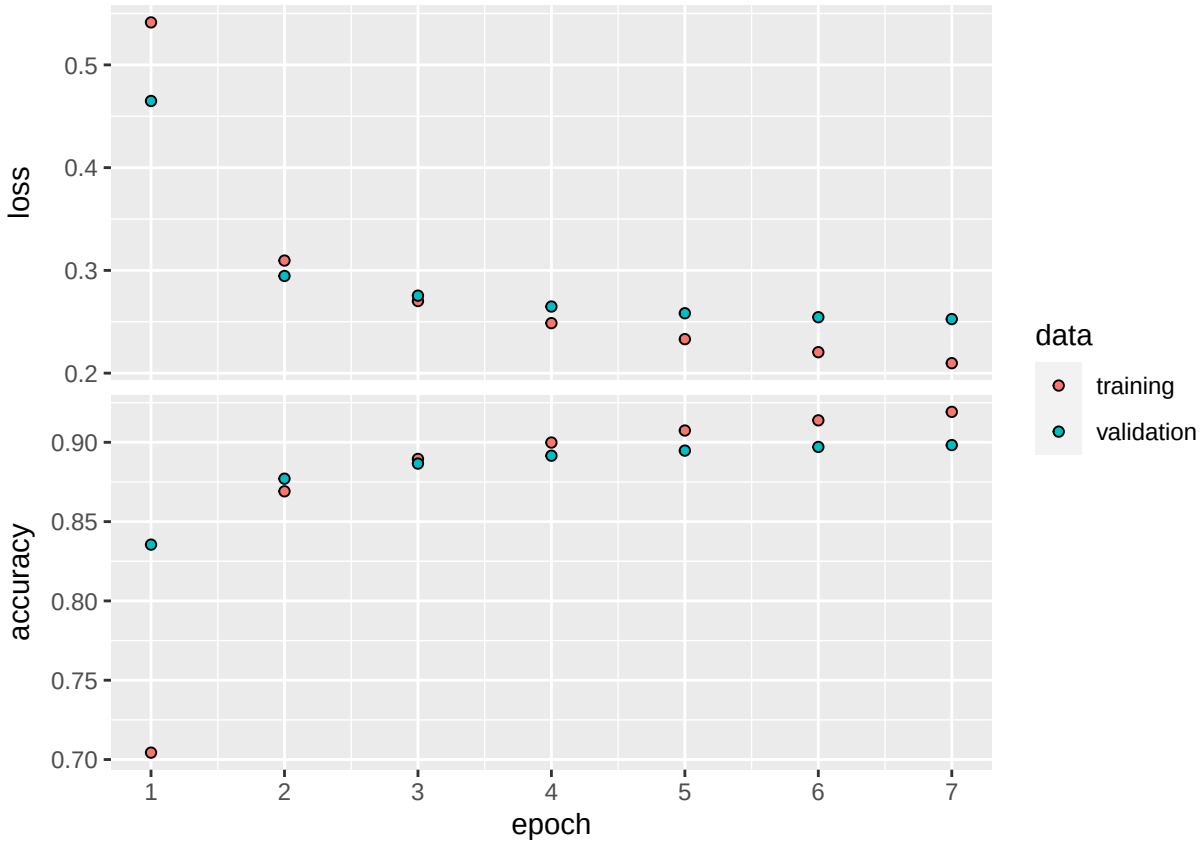

Figure 14: CNN Model Fit History using validation_data

Table 14: CNN Model Metrics using validation data

.metric	.estimator	.estimate
accuracy	binary	0.8982586
kap	binary	0.7911622

"CNN Model Confusion Matrix using validation_data" Figure 15

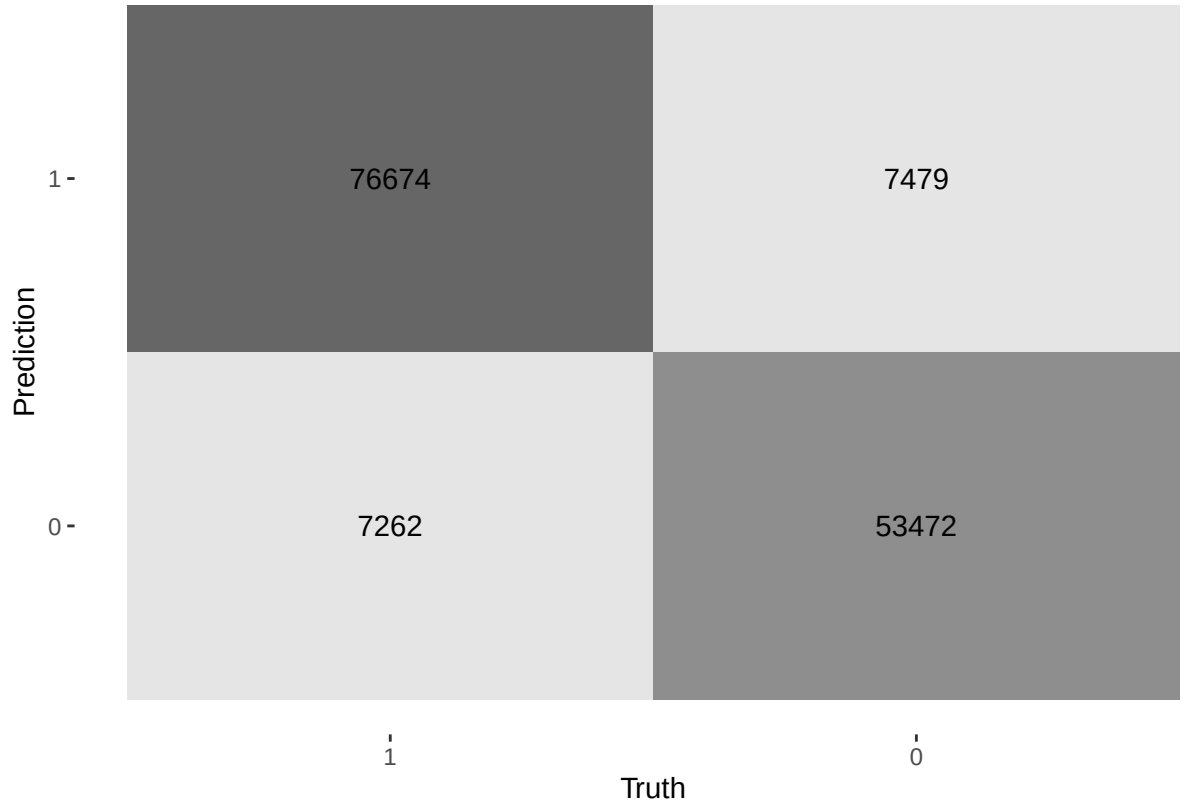

Figure 15: CNN Model Confusion Matrix using validation_data

"CNN Model ROC curve using validation_data" Figure 16

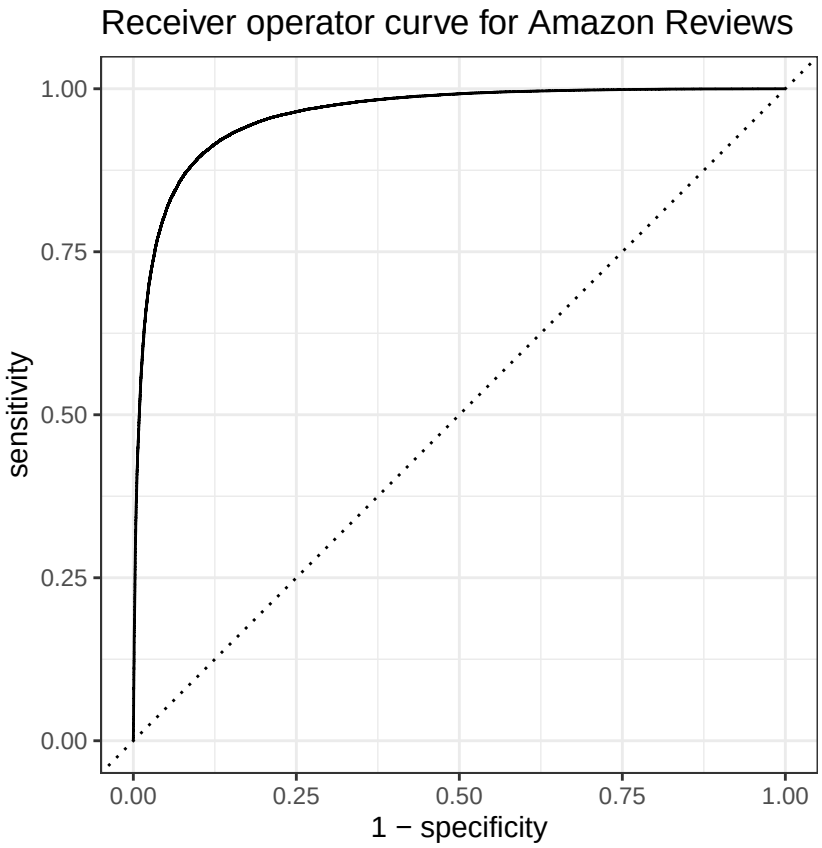

Figure 16: CNN Model ROC curve using validation_data

5.2 Results

CNN model results Table 15:

Table 15: CNN Model Results

Index	Method	Accuracy	Loss
1	BLM	0.8897064	NA
2	DNN	0.8960983	0.2790776
3	CNN	0.8982586	0.2526298

6 Model sepCNN

A convolutional neural network architecture based on Inception[32], but where Inception modules are replaced with depthwise separable convolutions. Also known as Xception.

A depthwise separable convolution (also abbreviated as separable convolution) factors a standard 3-D convolution into two separate convolution operations that are more computationally efficient: first, a depthwise convolution, with a depth of 1 (n x n x 1), and then second, a pointwise convolution, with length and width of 1 (1 x 1 x n).

To learn more, see Xception: Deep Learning with Depthwise Separable Convolutions[33].

6.1 A first sepCNN model

```
sep_cnn_model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_words + 1, output_dim = 16, input_length = max_length) %>%
    # layer_batch_normalization() %>%
layer_dropout(0.2) %>%
    layer_separable_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
    layer_separable_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
    layer_max_pooling_1d(pool_size = 2) %>%
    layer_separable_conv_1d(filter = 64, kernel_size = 5, activation = "relu") %>%
    layer_separable_conv_1d(filter = 64, kernel_size = 5, activation = "relu") %>%
    layer_global_average_pooling_1d() %>%
    layer_dropout(0.2) %>%
    # layer_dense(units = 64, activation = 'relu') %>%
layer_dense(units = 1, activation = "sigmoid")

sep_cnn_model
Model
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 30, 16)	320016
dropout_3 (Dropout)	(None, 30, 16)	0
separable_conv1d_3 (SeparableConv1D	(None, 26, 32)	624
separable_conv1d_2 (SeparableConv1D	(None, 22, 32)	1216
max_pooling1d_1 (MaxPooling1D)	(None, 11, 32)	0
separable_conv1d_1 (SeparableConv1D	(None, 7, 64)	2272
separable_conv1d (SeparableConv1D)	(None, 3, 64)	4480
global_average_pooling1d (GlobalAve	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0

[32] https://github.com/tensorflow/tpu/tree/master/models/experimental/inception
[33] https://arxiv.org/pdf/1610.02357.pdf

```
_________________________________________________________________________________
dense_8 (Dense)                      (None, 1)                         65
=================================================================================
Total params: 328,673
Trainable params: 328,673
Non-trainable params: 0
_________________________________________________________________________________
```

```
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6) opt <- optimizer_adam() opt <-
# optimizer_sgd()
opt <- optimizer_adam(lr = 1e-04, decay = 1e-06)
sep_cnn_model %>%
    compile(optimizer = opt, loss = "binary_crossentropy", metrics = c("accuracy"))
```

Here we use a callback[34].

keras::callback_early_stopping: Interrupt training when validation performance has stopped improving. patience: number of epochs with no improvement after which training will be stopped.

Try ??keras::callback_early_stopping

```
sep_cnn_val_history <- sep_cnn_model %>%
  fit(
    x = amazon_analysis,
    y = label_analysis,
    batch_size = 128,
    epochs = 20,
    initial_epoch = 0,
    validation_data = list(amazon_assess, label_assess),
    callbacks = list(callback_early_stopping(
        monitor='val_loss', patience=2)),
    verbose = 2
  )

sep_cnn_val_history
```

```
##
## Final epoch (plot to see history):
##         loss: 0.1894
##     accuracy: 0.9279
##     val_loss: 0.2417
## val_accuracy: 0.9052
```

"sepCNN Model Fit History using validation_data" Figure 17

See Table 16 for "sepCNN Model Predictions using validation data"

```
sep_cnn_val_res <- keras_predict(sep_cnn_model, amazon_assess, label_assess)
# head(sep_cnn_val_res)
kable(head(sep_cnn_val_res), format="simple", caption="sepCNN Model Predictions using validation data\\
```

[34] https://tensorflow.rstudio.com/guide/keras/guide_keras/#sts=Callbacks

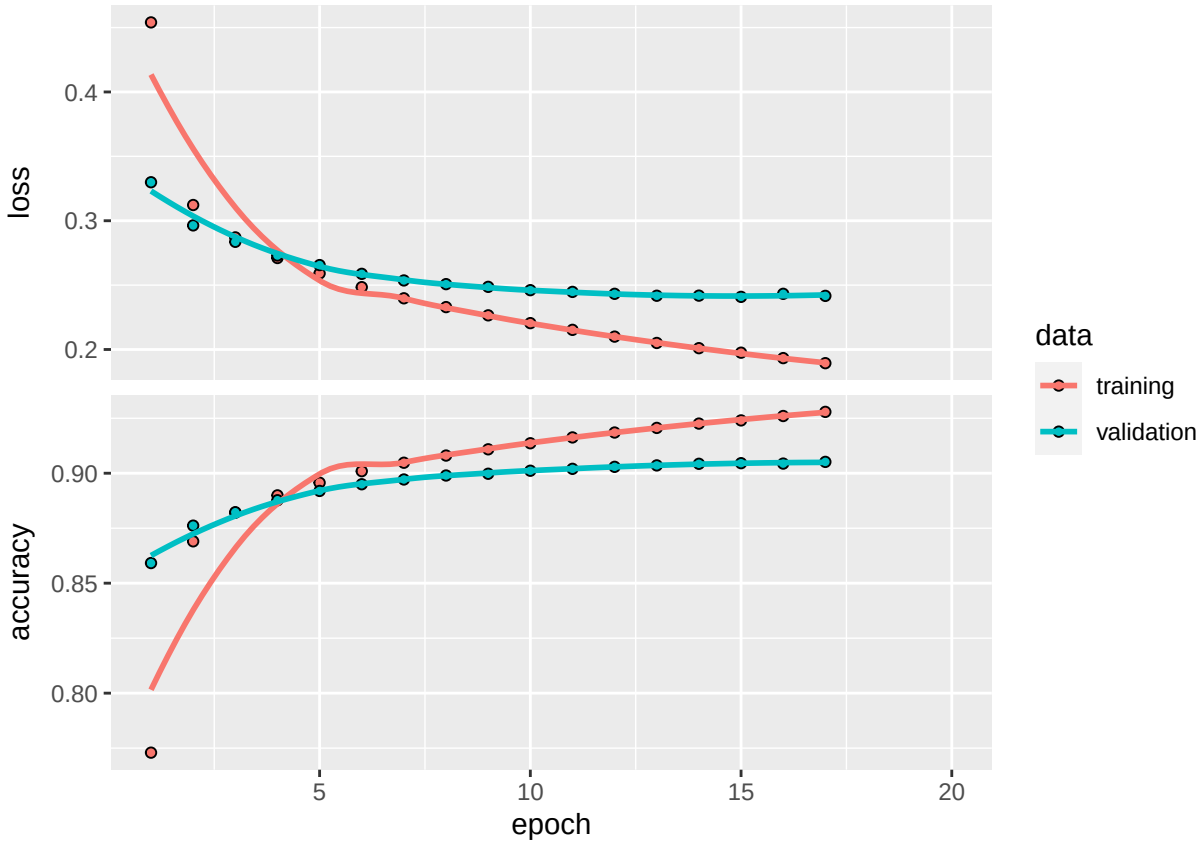

Figure 17: sepCNN Model Fit History using validation_data

Table 16: sepCNN Model Predictions using validation data

.pred_1	.pred_class	label
0.0000648	0	0
0.9813509	1	1
0.9909515	1	1
0.2639050	0	0
0.8420113	1	1
0.9995446	1	1

See Table 17 for "sepCNN Model Metrics using validation data"

```
m3 <- metrics(sep_cnn_val_res, label, .pred_class)
kable(m3, format="simple", caption="sepCNN Model Metrics using validation data\\label{tbl:sep_cnn_val_r
```

Table 17: sepCNN Model Metrics using validation data

.metric	.estimator	.estimate
accuracy	binary	0.9051675
kap	binary	0.8045877

"sepCNN Model Confusion Matrix using validation_data" Figure 18

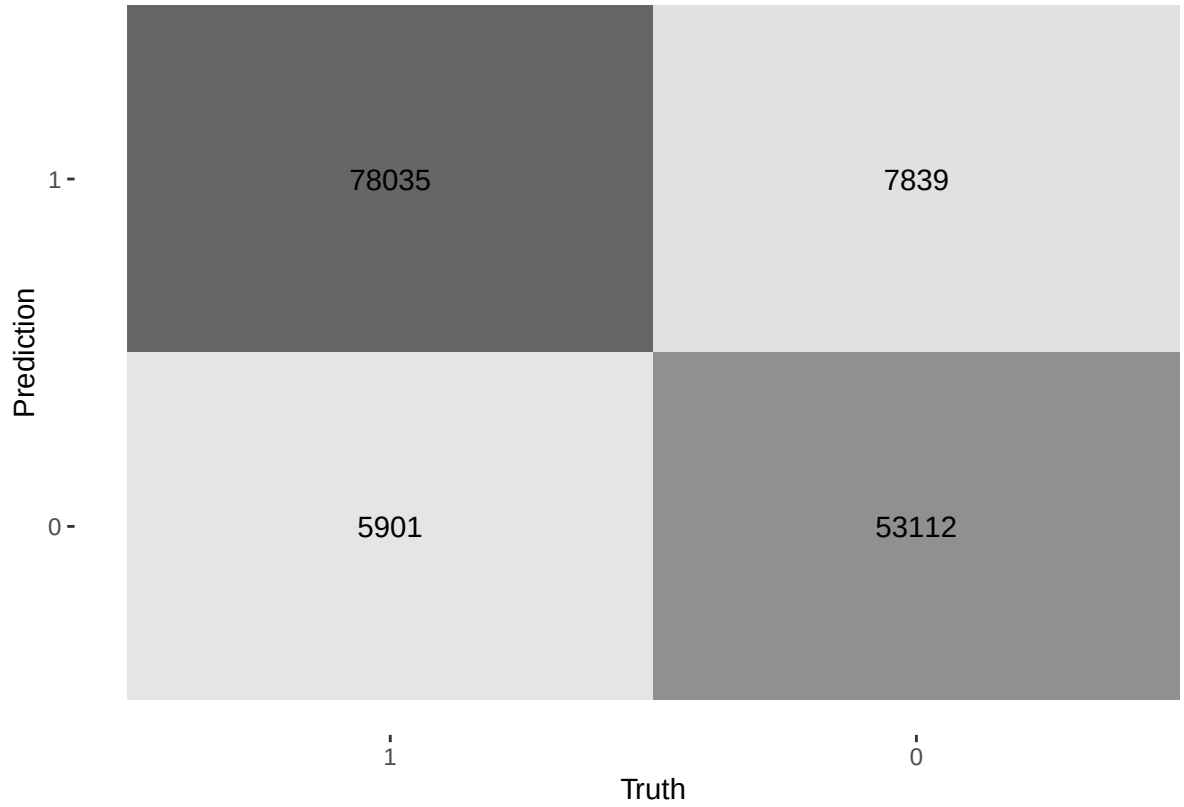

Figure 18: sepCNN Model Confusion Matrix using validation_data

"sepCNN Model ROC curve using validation_data" Figure 19

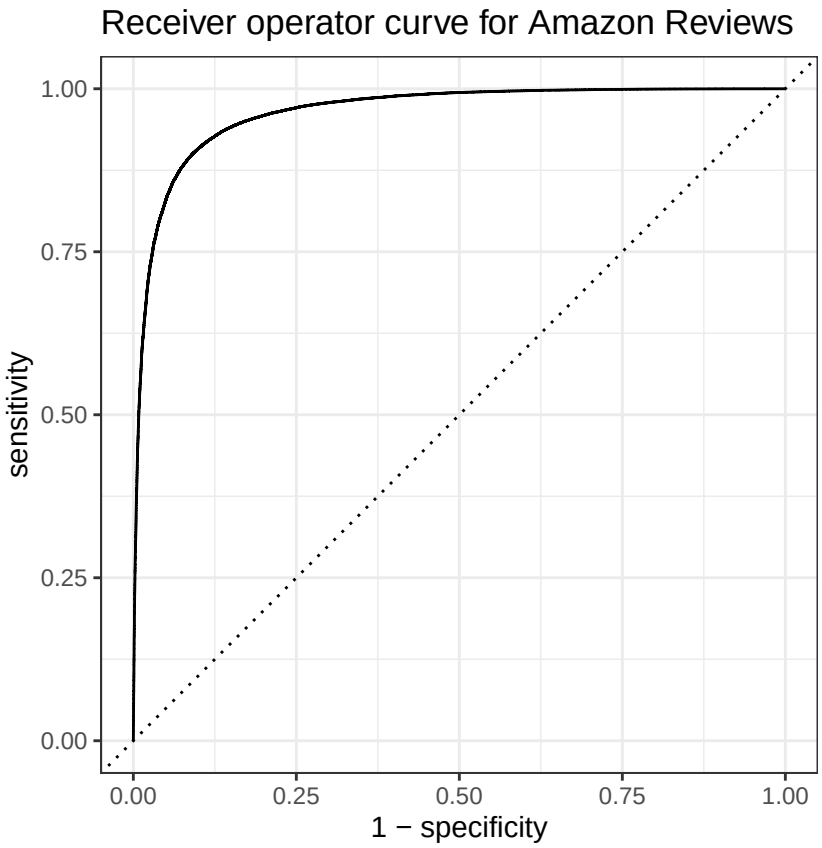

Figure 19: sepCNN Model ROC curve using validation_data

6.2 Results

sepCNN model results Table 18:

Table 18: sepCNN Model Results

Index	Method	Accuracy	Loss
1	BLM	0.8897064	NA
2	DNN	0.8960983	0.2790776
3	CNN	0.8982586	0.2526298
4	sepCNN	0.9051675	0.2408866

7 Model BERT

7.1 About BERT

In this model we will fine-tune BERT to perform sentiment analysis on our dataset.

BERT[35] and other Transformer encoder architectures have been wildly successful on a variety of tasks in NLP (natural language processing). They compute vector-space representations of natural language that are suitable for use in deep learning models. The BERT family of models uses the Transformer encoder architecture to process each token of input text in the full context of all tokens before and after, hence the name: Bidirectional Encoder Representations from Transformers.

BERT models are usually pre-trained on a large corpus of text, then fine-tuned for specific tasks.

7.2 References

Tensorflow[36] is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

The TensorFlow Hub[37] lets you search and discover hundreds of trained, ready-to-deploy machine learning models in one place.

Tensorflow for R[38] provides an R interface for Tensorflow.

7.3 Loading CSV data

Load CSV data from a file into a TensorFlow Dataset using tfdatasets.

7.4 Setup

```
library(keras)
library(tfdatasets)
library(reticulate)
library(tidyverse)
library(lubridate)
library(tfhub)

# A dependency of the preprocessing for BERT inputs pip install -q -U
# tensorflow-text
import("tensorflow_text")
Module(tensorflow_text)

# You will use the AdamW optimizer from tensorflow/models.  pip install -q
# tf-models-official to create AdamW optimizer
o_nlp <- import("official.nlp")

Sys.setenv(TFHUB_CACHE_DIR = "C:/Users/bijoor/.cache/tfhub_modules")
```

[35] https://arxiv.org/abs/1810.04805
[36] https://www.tensorflow.org/
[37] https://tfhub.dev/
[38] https://tensorflow.rstudio.com/

```
Sys.getenv("TFHUB_CACHE_DIR")
[1] "C:/Users/bijoor/.cache/tfhub_modules"
```

You could load this using read.csv, and pass the arrays to TensorFlow. If you need to scale up to a large set of files, or need a loader that integrates with TensorFlow and tfdatasets then use the make_csv_dataset function:

Now read the CSV data from the file and create a dataset.

7.5 Make datasets

```
train_file_path <- file.path("amazon_review_polarity_csv/amazon_train.csv")

batch_size <- 32

train_dataset <- make_csv_dataset(train_file_path, field_delim = ",", batch_size = batch_size,
    column_names = list("label", "title", "text"), label_name = "label", select_columns = list("label",
        "text"), num_epochs = 1)


train_dataset %>%
    reticulate::as_iterator() %>%
    reticulate::iter_next()  #%>%
[[1]]
OrderedDict([('text', <tf.Tensor: shape=(32,), dtype=string, numpy=
array([b'I am years old and have read many books like this one before I found that it said little while
       b'I would not recommend this flashlight to anyone who actually wants to be able to see in the da
       b'Has tons of game storage but I bought a starter pack initially which hardly held more than the
       b'An abortion Read other star reviews for more depth but please please do not read anything trans
       b'I have loved Wrinkle in Time since I first read it in grade school It is a wonderful book that
       b'I hate sounding like a commercial but this product is amazing It melted the mastic into a pool
       b'This book is unusual for a Holocaust book because none of the protagonists are wholly good or
       b'This product is strange First off every time i start a game I have to unplug the card to get a
       b'I used this figure for parts for a custom Darth Sion figure I made It actually makes a great Da
       b'I have many problems with this book First it is not a reprint of a old book but a photocopy of
       b'Had used it for about months and within the last months I was thinking to throw it to the ceme
       b'The newest remake of this movie doesn t do justice to the story This is an all time great and
       b'I have a nine foot tree The bottom three pieces needed to go in their own bag and the top two
       b'I have recently purchased this wine making book It contains many receipies for many diferent ty
       b'I think this is a very comprehensive report on reflux problems I have a sonwith Barrett s Esopl
       b'If I could rate this game lower than star I would You need an NVIDIA Video card to play this ga
       b'I really wanted to love these treats They re healthy and smell wonderful but my dogs will NOT
       b'Having read most of her books Agatha Christie still outwits me Very seldom one can guess the fi
       b'My lb jack russel destroyed two of these in hours of purchase don t waste ur money regular tenn
       b'this salad spinner does not work and I m surprised at so many positive reviews The old string
       b'i am one of paulo coelho s greatest fans when i saw this book in the bookstore couldnt think o
       b'The first time I turned the oven on it gave off a horrible chemical smell and set off the smok
       b'I am an old (electrical engineer graduation ) and new engineering student (just started a bioen
       b'Excellent product for the value Description should have listed that actual cleats were screw o
       b'I enjoyed Bryson s Walk in the Woods and recommend it frequently I wish I could say the same al
       b'I am not one to stuff into a import cd so I was pretty happy to find a Maaya Sakanoto cd that
       b'This book is so far out of date as to be unusable Has very little relationship with C Builder
       b'This is not your Blackmore s Purple Tommy Bolin brings a real spark of life to the beast and i
       b'If you have read evene a few pages of any book by thix Nixonite then you hve tapped into the b
```

```
       b'I bought the Scosche IPNRFC Wireless Remote mainly to use on my Jetski It makes it so much easi
       b'The primary issue with his unit is that the word thermal is misleading This unit will not heat
       b'Berry reaches for the literary high ground but he stumbles on endlessly repeated imagery and nu
      dtype=object)>)])

[[2]]
tf.Tensor([0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 1 0], shape=(32,), dtype=int32)
# reticulate::py_to_r()

# ----------------------

val_file_path <- file.path("amazon_review_polarity_csv/amazon_val.csv")

val_dataset <- make_csv_dataset(val_file_path, field_delim = ",", batch_size = batch_size,
    column_names = list("label", "title", "text"), label_name = "label", select_columns = list("label",
        "text"), num_epochs = 1)


val_dataset %>%
    reticulate::as_iterator() %>%
    reticulate::iter_next()
[[1]]
OrderedDict([('text', <tf.Tensor: shape=(32,), dtype=string, numpy=
array([b'Wish I d caught on earlier Everyone told me that the Harry Potter series was good I my expecta
       b'Jennifer Love Hewitt doesn t look a thing like Audrey Hepburn I truely believe that the produc
       b'Absolutely charming book great for all grade levels I usually run for the gory werewolfie or a
       b'This movie was made to be a fantasy martial arts movie and it delivers as promised I have a ve
       b'Aircraft physics cgi are horrible Planes don t fly that way And the German guy is of course dr
       b'I had never read Treasure Island but upon receiving my kindle and the fact this was a free rea
       b'I tend to agree with some minority reviewers that this book is boring and dull It has some int
       b'I read about halfway through this book and then I gave up I read James Turn of the Screw and Da
       b'Typical cavalry versus Indian oater Lots of action which is what I ask from a Western',
       b'This movie is pretty good It didn t require an indepth knowledge of the TV show to follow the m
       b'This is first time I have been compelled to write a bad review but frankly this movie was bad b
       b'The Third Secret is another entry in the Vatican intrigue genre It has the usual elements Vati
       b'I can t believe I wasted my money on this When I got the product I had before hand read some re
       b'The Spongebob Squarepants Movie had an unoriginal title but I LOVED the movie anyway I only lik
       b'Under the table surface there are two factory assembled bars used to connect the four legs Clea
       b'I read the comment by the reviewer from Montgomery Alabama back in September that the ASV would
       b'I ordered these beds because my grandkids were sleeping on the floor and couldnt afford real be
       b'I looked forward to seeing this movie but it was a huge letdown Besides the historical inaccura
       b'Bought this book by accident and now I love it It is easy to take to clinicals or even to look
       b'I went into this movie that it was going to be stupid But I didn t even find it funny Sure ther
       b'And these masters are Michael Bay and his computers And we have no one but ourselves to blame T
       b'I was surprised and delighted to see this remake After purchasing I was well disappointed I gue
       b'The whole album is amazing and hyper ballad has got to be one of the most beautiful songs ever
       b'I have a Samsung Blu Ray and the BR disc of the th Anniversary version of Dirty Dancing The pic
       b'I like family love movie very much From the view point of a father this movie is excellent In t
       b'Mine quite pumping water after cups I should have known better since my oldest son had sent his
       b'I am the author of this book It is no longer being printed I gave it only four stars because it
       b'This is the book that got me interested in reading mystery novels It s suspenseful thought prov
       b'I did not really care for this film though it did have a few funny moments a lot of the attempt
       b'This book had so much potential to be far more than what it was I agree with other customers wh
       b'I wear a size I took the info left by others and ordered a size They fit great Warm but not hot
```

```
       b'Mr Waltari s storytelling style comes forth in colorful detail and complex characters The mix
      dtype=object)>)])

[[2]]
tf.Tensor([1 0 1 1 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1], shape=(32,), dtype=int32)

# ------------------------------------

test_file_path <- file.path("amazon_review_polarity_csv/amazon_test.csv")


test_dataset <- make_csv_dataset(test_file_path, field_delim = ",", batch_size = batch_size,
    column_names = list("label", "title", "text"), label_name = "label", select_columns = list("label",
        "text"), num_epochs = 1)

test_dataset %>%
    reticulate::as_iterator() %>%
    reticulate::iter_next()
[[1]]
OrderedDict([('text', <tf.Tensor: shape=(32,), dtype=string, numpy=
array([b'The one star is for Liam s hair on the inside cover At least that still has it s dignity As fo
       b'The spreader is much larger and wider than I wanted I have found it useless for buttering brea
       b'This is so typical of H James writing very make that incredibly dense Wordy beyond belief At a
       b'I previous had a similar jar opener and have places and wanted to keep one in one place and th
       b'Wasn t sure what was wrong with my ice maker so I bought this kit Replaced just the main ice m
       b'This book is a watershed in human intellectual history In it Freud undermines the picture of m
       b'The game includes nowhere to store the black sheets for the screen The storage drawer jams eve
       b'This is a very good cd you should buy it My favourite songs are Why does my heart feel so bad
       b'I thought at first that the vacuum was extremely hard to push until I realized that the lowest
       b'There are many accolades and volumes of in depth analysis for this film so my review will be s
       b'This book did not move me at all The plot is unrealistic and the characters are shallow they d
       b'Aaliyah has definitely grown up The songs are about love hurt pain sex etc The songs are more
       b'Whoa boy I ve seen some pretty lame flicks last year but The Skulls definitely comes in as one
       b'In Flames and Soilwork are sold out so my only chance to find another album that blow my mind
       b'I guess I should preface this by saying that I m really not an Asian movie person I just don t
       b'OPENED THE BOX AND IT WAS NOT CARBON FIBER IT WAS CLEAR AMAZON NEEDS TO CHECK THEIR INVENTORY
       b'Danger Kitty has mass produced some of the worst music in the history of Pop To all of the BJ
       b'I am only years old but at least read a book a week When i first picked up this book i felt i
       b'I thought the book was very good It does get a little boring to me because of so much explicit
       b'This book was given to me I read it and placed it on my book shelf As a writer I have used the
       b'The first of this movie are spent developing the characters of Liu Xing and Prof Reiser and th
       b'The Nantucket Blend is one of my favorite k cup coffees that I ve made It s a medium blend cof
       b'My wife and I are planning a trip to Tuscany in May and ordered this video to get ideas about
       b'Okay if you re a special effects freak you might like this However there are much better films
       b'I checked this book out from my local La Leche League I m surprised they still have this book
       b'Bought this for wife as a stocking stuffer I m sure she ll enjoy backing her team without taki
       b'Recently bought the Sony SS B speakers As many have mentioned are a little larger than most bo
       b'Was Very Very disappointed Movie though brand new just out of the box it was stopping all the
       b'This is the worst movie you will ever watch no plot no story terrible casting and nothing to d
       b'I just got this unit and must say that I am disappointed It maybe because I was expecting it f
       b'This book is stupid the romance the vampires everything and why did she use that poem Amelia n
       b'My Xantrex XPower Powerpack Heavy Duty is not good for an emergency It is an emergency The uni
      dtype=object)>)])
```

```
[[2]]
tf.Tensor([0 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0], shape=(32,), dtype=int32)

rm(amazon_orig_train, amazon_orig_test, amazon_train, amazon_val)
Warning in rm(amazon_orig_train, amazon_orig_test, amazon_train, amazon_val):
object 'amazon_orig_train' not found
Warning in rm(amazon_orig_train, amazon_orig_test, amazon_train, amazon_val):
object 'amazon_orig_test' not found
Warning in rm(amazon_orig_train, amazon_orig_test, amazon_train, amazon_val):
object 'amazon_val' not found
rm(ids_train, train_file_path, test_file_path, val_file_path)
Warning in rm(ids_train, train_file_path, test_file_path, val_file_path): object
'ids_train' not found
```

7.6 The preprocessing model

Text inputs need to be transformed to numeric token ids and arranged in several Tensors before being input to BERT. TensorFlow Hub provides a matching preprocessing model for the BERT models, which implements this transformation using TF ops from the TF.text library.

The preprocessing model must be the one referenced by the documentation of the BERT model, which you can read at the URL bert_en_uncased_preprocess[39]

```
bert_preprocess_model <- layer_hub(handle = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3"
    trainable = FALSE, name = "preprocessing")
```

7.7 Using the BERT model

The BERT models return a map with 3 important keys: pooled_output, sequence_output, encoder_outputs:

"pooled_output" represents each input sequence as a whole. The shape is [batch_size, H]. You can think of this as an embedding for the entire Amazon review.

"sequence_output" represents each input token in the context. The shape is [batch_size, seq_length, H]. You can think of this as a contextual embedding for every token in the Amazon review.

"encoder_outputs" are the intermediate activations of the L Transformer blocks. outputs["encoder_outputs"][i] is a Tensor of shape [batch_size, seq_length, 1024] with the outputs of the i-th Transformer block, for 0 <= i < L. The last value of the list is equal to sequence_output.

For the fine-tuning you are going to use the pooled_output array.

For more information about the base model's input and output you can follow the model's URL at small_bert/bert_en_uncased_L-4_H-512_A-8[40]

```
bert_model <- layer_hub(handle = "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8
    trainable = TRUE, name = "BERT_encoder")
```

7.8 Define your model

We will create a very simple fine-tuned model, with the preprocessing model, the selected BERT model, one Dense and a Dropout layer.

```
input <- layer_input(shape = shape(), dtype = "string", name = "text")

output <- input %>%
    bert_preprocess_model() %>%
    bert_model %$%
    pooled_output %>%
    layer_dropout(0.1) %>%
    # layer_dense(units = 16, activation = 'relu') %>%
layer_dense(units = 1, activation = "sigmoid", name = "classifier")

# summary(model)

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

[39] https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
[40] https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/2

Loss function

Since this is a binary classification problem and the model outputs a probability (a single-unit layer), you'll use "binary_crossentropy" loss function.

Optimizer

For fine-tuning, let's use the same optimizer that BERT was originally trained with: the "Adaptive Moments" (Adam). This optimizer minimizes the prediction loss and does regularization by weight decay (not using moments), which is also known as AdamW[41].

We will use the AdamW optimizer from tensorflow/models[42].

For the learning rate (init_lr), you will use the same schedule as BERT pre-training: linear decay of a notional initial learning rate, prefixed with a linear warm-up phase over the first 10% of training steps (num_warmup_steps). In line with the BERT paper, the initial learning rate is smaller for fine-tuning (best of 5e-5, 3e-5, 2e-5).

```
epochs = 5
steps_per_epoch <- 2e+06
num_train_steps <- steps_per_epoch * epochs
num_warmup_steps <- as.integer(0.1 * num_train_steps)

init_lr <- 3e-05
opt <- o_nlp$optimization$create_optimizer(init_lr = init_lr, num_train_steps = num_train_steps,
    num_warmup_steps = num_warmup_steps, optimizer_type = "adamw")

model %>%
    compile(loss = "binary_crossentropy", optimizer = opt, metrics = "accuracy")

summary(model)
Model: "model"
__________________________________________________________________________________
Layer (type)                Output Shape      Param #  Connected to
==================================================================================
text (InputLayer)           [(None,)]         0
__________________________________________________________________________________
preprocessing (KerasLayer {'input_mask': (N 0        text[0][0]
__________________________________________________________________________________
BERT_encoder (KerasLayer) {'default': (None 28763649 preprocessing[0][0]
                                                     preprocessing[0][1]
                                                     preprocessing[0][2]
__________________________________________________________________________________
dropout_4 (Dropout)         (None, 512)       0        BERT_encoder[0][5]
__________________________________________________________________________________
classifier (Dense)          (None, 1)         513      dropout_4[0][0]
==================================================================================
Total params: 28,764,162
Trainable params: 28,764,161
Non-trainable params: 1
__________________________________________________________________________________
```

REMEMBER to change tr_count back to 10000 for better training.

[41] https://arxiv.org/abs/1711.05101
[42] https://github.com/tensorflow/models

Try a sample/subset to train/test code, and to reduce training time due to resource constraints use a smaller tr_count below.

```
# 10000 will take approx 40 mins per epoch on my gpu/mem etc 1000 will take
# approx 4 mins per epoch on my gpu/mem etc

tr_count <- 10000
take_tr <- 0.8 * tr_count
train_slice <- train_dataset %>%
    dataset_shuffle_and_repeat(buffer_size = take_tr * batch_size) %>%
    dataset_take(take_tr)

take_val <- 0.2 * tr_count
val_slice <- val_dataset %>%
    dataset_shuffle_and_repeat(buffer_size = take_val * batch_size) %>%
    dataset_take(take_val)


epochs <- 5
seed = 42

history <- model %>%
    fit(train_slice, epochs = epochs, validation_data = val_slice, initial_epoch = 0,
        verbose = 2)
```

"BERT Model Fit History using validation_data slice" Figure 20

Evaluate the model

Let's see how the model performs. Two values will be returned. Loss (a number which represents the error, lower values are better), and accuracy.

Takes too long, so skipping it for now. Using test_slice instead.

```
model %>%
    evaluate(test_dataset)
```

Using test_slice instead.

```
test_slice <- test_dataset %>%
    dataset_take(100)

model %>%
    evaluate(test_slice)
     loss  accuracy
0.2556886 0.9012500
```

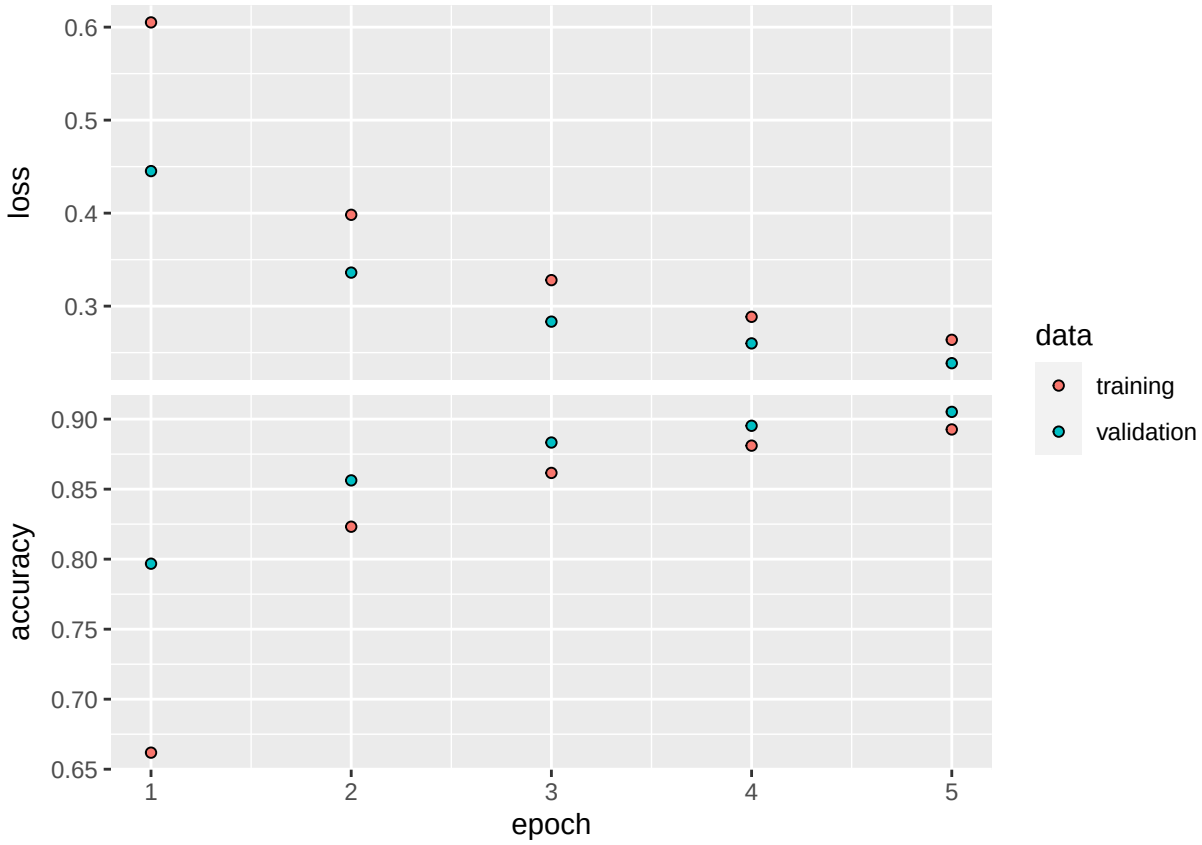

Figure 20: BERT Model Fit History using validation_data slice

7.9 Results

BERT model results Table 19:

Table 19: BERT Model Results

Index	Method	Accuracy	Loss
1	BLM	0.8897064	NA
2	DNN	0.8960983	0.2790776
3	CNN	0.8982586	0.2526298
4	sepCNN	0.9051675	0.2408866
5	BERT	0.9051719	0.2386644

```
# Stop the clock
# proc.time() - ptm
Sys.time()
```

```
## [1] "2021-07-28 05:13:36 EDT"
```

8 Conclusion

In this project, we attempted to significantly simplify the process of selecting a text classification model. For a given dataset, our goal was to find the algorithm that achieved close to maximum accuracy while minimizing computation time required for training.

CNNs are a type of neural network that can learn local spatial patterns. They essentially perform feature extraction, which can then be used efficiently in later layers of a network. Their simplicity and fast running time, compared to other models, makes them excellent candidates for supervised models for text.

Based on our results and inspite of using only a fraction of our data due to (my) resource limitations, we agree with Google and conclude that sepCNN's and/or BERT helped us achieve our goal of simplicity, minimum compute time and maximum accuracy.

As of now, my future attempts in ML will be in NLP related activities.

---

9 Appendix: All code for this report

```
knitr::knit_hooks$set(time_it = local({
  now <- NULL
  function(before, options) {
    if (before) {
      # record the current time before each chunk
      now <<- Sys.time()
    } else {
      # calculate the time difference after a chunk
      res <- difftime(Sys.time(), now)
      # return a character string to show the time
      # paste("Time for this code chunk to run:", res)
      paste("Time for the chunk", options$label, "to run:", res)
    }
  }
}))

# knit_hooks$get("inline")
# knitr::opts_chunk$set(fig.pos = "!H", out.extra = "")
knitr::opts_chunk$set(echo = TRUE,
                      fig.path = "figures/")

# Beware, using the "time_it" hook messes up fig.cap, \label, \ref
# knitr::opts_chunk$set(time_it = TRUE)
#knitr::opts_chunk$set(eval = FALSE)
options(tinytex.verbose = TRUE)

# set pandoc stack size
stack_size <- getOption("pandoc.stack.size", default = "100000000")
args <- c(c("+RTS", paste0("-K", stack_size), "-RTS"), args)
# library(dplyr)
# library(tidyr)
# library(purrr)
# library(readr)
library(tidyverse)
library(textrecipes)
library(tidymodels)
library(tidytext)
library(ngram)
library(keras)
library(stopwords)

# Used in Baseline model
library(hardhat)

# BERT setup in its own section
# library(keras)
# library(tfdatasets)
# library(reticulate)
# library(tidyverse)
# library(lubridate)
# library(tfhub)
```

```r
# import("tensorflow_text")
# o_nlp <- import("official.nlp")
#
# Sys.setenv(TFHUB_CACHE_DIR="C:/Users/bijoor/.cache/tfhub_modules")
# Sys.getenv("TFHUB_CACHE_DIR")

set.seed(234)

# Start the clock!
# ptm <- proc.time()
Sys.time()
  library(ggplot2)
  library(kableExtra)
untar("amazon_review_polarity_csv.tar.gz",list=TRUE)  ## check contents
untar("amazon_review_polarity_csv.tar.gz")
train_file_path <- file.path("amazon_review_polarity_csv/train.csv")

test_file_path <- file.path("amazon_review_polarity_csv/test.csv")

# read data, ensure "utf-8" encoding, add column names, exclude rows with missing values(NA)
amazon_orig_train <- readr::read_csv(
  train_file_path,
  # skip = 0,
  col_names = c("label", "title", "text"),
  locale = locale(encoding = "UTF-8")) %>% na.omit()

# change labels from (1,2) to (0,1) - easier for binary classification
amazon_orig_train$label[amazon_orig_train$label==1] <- 0
amazon_orig_train$label[amazon_orig_train$label==2] <- 1

# removed numbers as they were too many and did not contribute any info

# amazon_orig_train$text <- str_replace_all(amazon_orig_train$text,"[^([[:alnum:]_])]"," ") %>% trimws(
#
# amazon_orig_train$title <- str_replace_all(amazon_orig_train$title,"[^([[:alnum:]_])]"," ") %>% trimw

# remove leading/trailing whitespace (trimws)
# trim whitespace from a string (str_squish)
# replace non alphabet chars with space

amazon_orig_train$text <- str_replace_all(amazon_orig_train$text,"[^([[:alpha:]_])]"," ") %>% trimws() %

amazon_orig_train$title <- str_replace_all(amazon_orig_train$title,"[^([[:alpha:]_])]"," ") %>% trimws()

# create a validation set for training purposes
ids_train <- sample.int(nrow(amazon_orig_train), size = 0.8*nrow(amazon_orig_train))
amazon_train <- amazon_orig_train[ids_train,]
amazon_val <- amazon_orig_train[-ids_train,]

head(amazon_train)

# save cleaned up data for later use
write_csv(amazon_train,"amazon_review_polarity_csv/amazon_train.csv", col_names = TRUE)
write_csv(amazon_val,"amazon_review_polarity_csv/amazon_val.csv", col_names = TRUE)
```

```r
# ------------------------------------------------
# read data, ensure "utf-8" encoding, add column names, exclude rows with missing values(NA)
amazon_orig_test <- readr::read_csv(
  test_file_path,
  # skip = 0,
  col_names = c("label", "title", "text"),
  locale = locale(encoding = "UTF-8")) %>% na.omit()

# change labels from (1,2) to (0,1) - easier for binary classification
amazon_orig_test$label[amazon_orig_test$label==1] <- 0
amazon_orig_test$label[amazon_orig_test$label==2] <- 1

# remove leading/trailing whitespace (trimws)
# trim whitespace from a string (str_squish)
# replace non alphabet chars with space

amazon_orig_test$text <- str_replace_all(amazon_orig_test$text,"[^([[:alpha:]_])]"," ") %>% trimws() %>%

amazon_orig_test$title <- str_replace_all(amazon_orig_test$title,"[^([[:alpha:]_])]"," ") %>% trimws() %

# amazon_orig_test$text <- str_replace_all(amazon_orig_test$text,"[^([[:alnum:]_])]"," ") %>% trimws() %
#
# amazon_orig_test$title <- str_replace_all(amazon_orig_test$title,"[^([[:alnum:]_])]"," ") %>% trimws()

head(amazon_orig_test)

# save cleaned up data for later use
write_csv(amazon_orig_test,"amazon_review_polarity_csv/amazon_test.csv", col_names = TRUE)

rm(amazon_orig_train, amazon_orig_test)
rm(ids_train, test_file_path, train_file_path)

# free unused R memory
gc()
#### To be deleted later
amazon_train <- readr::read_csv("amazon_review_polarity_csv/amazon_train.csv")
glimpse(amazon_train)
# head(amazon_train)
kable(amazon_train[1:10,], "latex", escape=FALSE, booktabs=TRUE, linesep="", caption="Amazon Train  data
    kable_styling(latex_options=c("HOLD_position"), font_size=6)
  # kable_styling(full_width = F)
unique(amazon_train$label)
(num_samples <- nrow(amazon_train))
(num_classes <- length(unique(amazon_train$label)))
# Pretty Balanced classes
(num_samples_per_class <- amazon_train %>% count(label))
# break up the strings in each row by " "
temp <- strsplit(amazon_train$text, split=" ")

# sapply(temp[c(1:3)], length)
# count the number of words as the length of the vectors
amazon_train_text_wordCount <- sapply(temp, length)
```

```
(mean_num_words_per_sample <- mean(amazon_train_text_wordCount))

(median_num_words_per_sample <- median(amazon_train_text_wordCount))
# Frequency distribution of words(ngrams)
train_words <- amazon_train %>% unnest_tokens(word, text) %>% count(word,sort = TRUE)

total_words <- train_words %>%
    summarize(total = sum(n))

# Zipfs law states that the frequency that a word appears is inversely proportional to its rank.
train_words <- train_words %>%
    mutate(total_words) %>%
     mutate(rank = row_number(),
         `term frequency` = n/total)
# head(train_words)
kable(train_words[1:10,], "latex", escape=FALSE, booktabs=TRUE, linesep="", caption="Frequency distribu
    # kable_styling(latex_options=c("HOLD_position"), font_size=6)
train_words %>%
  top_n(25, n) %>%
  ggplot(aes(reorder(word,n),n)) +
  geom_col(binwidth = 1, alpha = 0.8) +
   coord_flip() +
  labs(y="n - Frequency distribution of words(ngrams)",
       x="Top 25 words")
length(stopwords(source = "smart"))
length(stopwords(source = "snowball"))
length(stopwords(source = "stopwords-iso"))
mystopwords <- c("s", "t", "m", "ve", "re", "d", "ll")

# Frequency distribution of words(ngrams)
train_words_sw <- amazon_train %>% unnest_tokens(word, text) %>%
    anti_join(get_stopwords(source = "stopwords-iso"))%>%
    filter(!(word %in% mystopwords)) %>%
    count(word,sort = TRUE)

total_words_sw <- train_words_sw %>%
    summarize(total = sum(n))

# Zipfs law states that the frequency that a word appears is inversely proportional to its rank.

train_words_sw <- train_words_sw %>%
    mutate(total_words_sw) %>%
     mutate(rank = row_number(),
         `term frequency` = n/total)
# head(train_words_sw)
kable(train_words_sw[1:10,], "latex", escape=FALSE, booktabs=TRUE, linesep="", caption="Frequency distri
    # kable_styling(latex_options=c("HOLD_position"), font_size=6)
train_words_sw %>%
  top_n(25, n) %>%
  ggplot(aes(reorder(word,n),n)) +
  geom_col(binwidth = 1, alpha = 0.8) +
   coord_flip() +
  labs(y="n - Frequency distribution of words(ngrams) excluding stopwords",
       x="Top 25 words")
```

```r
# 3. If the ratio is greater than 1500, tokenize the text as
# sequences and use a sepCNN model
#    see above

(S_W_ratio <- num_samples / median_num_words_per_sample)
amazon_train %>%
  mutate(n_words = tokenizers::count_words(text)) %>%
  ggplot(aes(n_words)) +
  geom_bar() +
  labs(x = "Number of words per review text",
       y = "Number of review texts")
amazon_train %>%
  mutate(n_words = tokenizers::count_words(title)) %>%
  ggplot(aes(n_words)) +
  geom_bar() +
  labs(x = "Number of words per review title",
       y = "Number of review titles")
amazon_train %>%
  group_by(label) %>%
  mutate(n_words = tokenizers::count_words(text)) %>%
  ggplot(aes(n_words)) +
  # ggplot(aes(nchar(text))) +
  geom_histogram(binwidth = 1, alpha = 0.8) +
  facet_wrap(~ label, nrow = 1) +
  labs(x = "Number of words per review text by label",
       y = "Number of reviews")
amazon_train %>%
  group_by(label) %>%
  mutate(n_words = tokenizers::count_words(title)) %>%
  ggplot(aes(n_words)) +
  # ggplot(aes(nchar(title))) +
  geom_histogram(binwidth = 1, alpha = 0.8) +
  facet_wrap(~ label, nrow = 1) +
  labs(x = "Number of words per review title by label",
       y = "Number of reviews")
amazon_subset_train <- amazon_train %>% select(-title) %>%
  mutate(n_words = tokenizers::count_words(text)) %>%
  filter((n_words < 35) & (n_words > 5)) %>% select(-n_words)

dim(amazon_subset_train)
# head(amazon_subset_train)
kable(amazon_subset_train[1:10,], "latex", escape=FALSE, booktabs=TRUE, linesep="", caption="Sample/Subs
    # kable_styling(latex_options=c("HOLD_position"), font_size=6)
# Free computer resources
rm(amazon_train, amazon_val, amazon_train_text_wordCount,num_samples_per_class, temp, total_words, trai
rm(mean_num_words_per_sample, median_num_words_per_sample, num_classes, num_samples, S_W_ratio)
gc()

# save(amazon_subset_train)
write_csv(amazon_subset_train,"amazon_review_polarity_csv/amazon_subset_train.csv", col_names = TRUE)

amazon_train <- amazon_subset_train

amazon_train <- amazon_train %>%
```

```
  mutate(label = as.factor(label))

# amazon_val <- amazon_train %>%
#   mutate(label = as.factor(label))
set.seed(1234)

amazon_split <- amazon_train %>% initial_split()

amazon_train <- training(amazon_split)
amazon_test <- testing(amazon_split)

set.seed(123)
amazon_folds <- vfold_cv(amazon_train)
# amazon_folds
# library(textrecipes)

amazon_rec <- recipe(label ~ text, data = amazon_train) %>%
  step_tokenize(text) %>%
  step_tokenfilter(text, max_tokens = 5e3) %>%
  step_tfidf(text)

amazon_rec
lasso_spec <- logistic_reg(penalty = tune(), mixture = 1) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

lasso_spec
library(hardhat)
sparse_bp <- default_recipe_blueprint(composition = "dgCMatrix")
lambda_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 20)
lambda_grid
amazon_wf <- workflow() %>%
  add_recipe(amazon_rec, blueprint = sparse_bp) %>%
  add_model(lasso_spec)

amazon_wf
set.seed(2020)
lasso_rs <- tune_grid(
  amazon_wf,
  amazon_folds,
  grid = lambda_grid,
  control = control_resamples(save_pred = TRUE)
)

# lasso_rs
m_lm <- collect_metrics(lasso_rs)
kable(m_lm, format = "simple", caption="Lasso Metrics\\label{tbl:lasso_metrics}")
m_blr <- show_best(lasso_rs, "roc_auc")
kable(m_blr, format = "simple", caption="Best Lasso ROC\\label{tbl:best_lasso_roc}")
m_bla <- show_best(lasso_rs, "accuracy")
kable(m_bla, format = "simple", caption="Best Lasso Accuracy\\label{tbl:best_lasso_acc}")
autoplot(lasso_rs) +
  labs(
    title = "Lasso model performance across regularization penalties",
```

```
    subtitle = "Performance metrics can be used to identify the best penalty"
  )
m_lp <- collect_predictions(lasso_rs)
kable(head(m_lp), format = "simple", caption="Lasso Predictions\\label{tbl:lasso_predictions}")
m_lp %>%
  # mutate(.pred_class=as.numeric(levels(.pred_class)[.pred_class])) %>%
  group_by(id) %>%
  roc_curve(truth = label, .pred_0) %>%
  autoplot() +
  labs(
    color = NULL,
    title = "ROC curve for Lasso model Label 0",
    subtitle = "Each resample fold is shown in a different color"
  )
m_lp %>%
  group_by(id) %>%
  roc_curve(truth = label, .pred_1) %>%
  autoplot() +
  labs(
    color = NULL,
    title = "ROC curve for Lasso model Label 1",
    subtitle = "Each resample fold is shown in a different color"
  )
# Best ROC_AUC
blm_best_roc <- max(m_blr$mean)

# Best Accuracy
blm_best_acc <- max(m_bla$mean)
results_table <- tibble(Index = "1", Method = "BLM", Accuracy = blm_best_acc, Loss = NA)

kable(results_table, "simple",caption="Baseline Linear Model Results\\label{tbl:blm_results_table}")
# %>%
#     kable_styling(latex_options=c("HOLD_position"), font_size=7)
rm(amazon_folds, amazon_rec, amazon_split, amazon_test, amazon_train, amazon_wf, lambda_grid, lasso_rs,

gc()

amazon_subset_train <- readr::read_csv("amazon_review_polarity_csv/amazon_subset_train.csv")

amazon_train <- amazon_subset_train

max_words <- 2e4
max_length <- 30
mystopwords <- c("s", "t", "m", "ve", "re", "d", "ll")

amazon_rec <- recipe(~ text, data = amazon_subset_train) %>%
  step_text_normalization(text) %>%
  step_tokenize(text) %>%
  step_stopwords(text,
                 stopword_source = "stopwords-iso",
                 custom_stopword_source = mystopwords) %>%
  step_tokenfilter(text, max_tokens = max_words) %>%
  step_sequence_onehot(text, sequence_length = max_length)
```

```
amazon_rec
amazon_prep <-  prep(amazon_rec)

amazon_subset_train <- bake(amazon_prep, new_data = NULL, composition = "matrix")
dim(amazon_subset_train)
amazon_prep %>% tidy(5) %>% head(10)
# library(keras)
# use_python(python = "/c/Users/bijoor/.conda/envs/tensorflow-python/python.exe", required = TRUE)
# use_condaenv(condaenv = "tensorflow-python", required = TRUE)

dense_model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words + 1,
                  output_dim = 12,
                  input_length = max_length) %>%
  layer_flatten() %>%
  layer_layer_normalization() %>%
  # layer_dropout(0.1) %>%
  layer_dense(units = 64) %>%
  # layer_activation_leaky_relu() %>%
  layer_activation_relu() %>%
  layer_dense(units = 1, activation = "sigmoid")

dense_model
# opt <- optimizer_adam(lr = 0.0001, decay = 1e-6)
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6)
opt <- optimizer_sgd()
dense_model %>% compile(
  optimizer = opt,
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
dense_history <- dense_model %>%
  fit(
    x = amazon_subset_train,
    y = amazon_train$label,
    batch_size = 1024,
    epochs = 50,
    initial_epoch = 0,
    validation_split = 0.20,
    verbose = 2
  )

dense_history

plot(dense_history)
set.seed(234)
amazon_val_eval <- validation_split(amazon_train, strata = label)
# amazon_val_eval << I am getting a pandoc stack error printing this
amazon_analysis <- bake(amazon_prep, new_data = analysis(amazon_val_eval$splits[[1]]),
                        composition = "matrix")
dim(amazon_analysis)
amazon_assess <- bake(amazon_prep, new_data = assessment(amazon_val_eval$splits[[1]]),
                      composition = "matrix")
dim(amazon_assess)
```

```
label_analysis <- analysis(amazon_val_eval$splits[[1]]) %>% pull(label)
label_assess <- assessment(amazon_val_eval$splits[[1]]) %>% pull(label)
dense_model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words + 1,
                  output_dim = 12,
                  input_length = max_length) %>%
  layer_flatten() %>%
  layer_layer_normalization() %>%
  layer_dropout(0.5) %>%
  layer_dense(units = 64) %>%
  layer_activation_relu() %>%
  layer_dropout(0.5) %>%
  layer_dense(units = 128) %>%
  layer_activation_relu() %>%
  layer_dense(units = 128) %>%
  layer_activation_relu() %>%
  layer_dense(units = 1, activation = "sigmoid")

opt <- optimizer_adam(lr = 0.0001, decay = 1e-6)
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6)
# opt <- optimizer_sgd()
dense_model %>% compile(
  optimizer = opt,
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
val_history <- dense_model %>%
  fit(
    x = amazon_analysis,
    y = label_analysis,
    batch_size = 2048,
    epochs = 20,
    validation_data = list(amazon_assess, label_assess),
    verbose = 2
  )

val_history
plot(val_history)
keras_predict <- function(model, baked_data, response) {
  predictions <- predict(model, baked_data)[, 1]
  tibble(
    .pred_1 = predictions,
    .pred_class = if_else(.pred_1 < 0.5, 0, 1),
    label = response) %>%
    mutate(across(c(label, .pred_class),
                  ~ factor(.x, levels = c(1, 0))))
}
val_res <- keras_predict(dense_model, amazon_assess, label_assess)
# head(val_res)
kable(head(val_res), format="simple", caption="DNN Model 2 Predictions using validation data\\label{tbl
m1 <- metrics(val_res, label, .pred_class)
kable(m1, format = "simple", caption="DNN Model 2 Metrics using Validation data\\label{tbl:val_res_metr
val_res %>%
  conf_mat(label, .pred_class) %>%
```

```
  autoplot(type = "heatmap")
val_res %>%
  roc_curve(truth = label, .pred_1) %>%
  autoplot() +
  labs(
    title = "Receiver operator curve for Amazon Reviews"
  )
# Best DNN accuracy
dnn_best_acc <- max(val_history$metrics$val_accuracy)

# Lowest DNN Loss
dnn_lowest_loss <- min(val_history$metrics$val_loss)
results_table <- bind_rows(results_table,
                           tibble(Index = "2",
                                  Method = "DNN",
                                  Accuracy = dnn_best_acc,
                                  Loss = dnn_lowest_loss))

kable(results_table, "simple",caption="DNN Model Results\\label{tbl:dnn_results_table}")
# %>%
#     kable_styling(latex_options=c("HOLD_position"), font_size=7)
simple_cnn_model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words + 1, output_dim = 16,
                  input_length = max_length) %>%
  layer_batch_normalization() %>%
  layer_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_conv_1d(filter = 64, kernel_size = 3, activation = "relu") %>%
  layer_global_max_pooling_1d() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

simple_cnn_model
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6)
# opt <- optimizer_adam()
# opt <- optimizer_sgd()
opt <- optimizer_adam(lr = 0.0001, decay = 1e-6)
simple_cnn_model %>% compile(
  optimizer = opt,
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
simple_cnn_val_history <- simple_cnn_model %>%
  fit(
    x = amazon_analysis,
    y = label_analysis,
    batch_size = 1024,
    epochs = 7,
    initial_epoch = 0,
    validation_data = list(amazon_assess, label_assess),
    verbose = 2
  )

simple_cnn_val_history
```

```r
plot(simple_cnn_val_history)
simple_cnn_val_res <- keras_predict(simple_cnn_model, amazon_assess, label_assess)
# head(simple_cnn_val_res)
kable(head(simple_cnn_val_res), format="simple", caption="CNN Model Predictions using validation data\\l
m2 <- metrics(simple_cnn_val_res, label, .pred_class)
kable(m2, format="simple", caption="CNN Model Metrics using validation data\\label{tbl:simple_cnn_val_r
simple_cnn_val_res %>%
  conf_mat(label, .pred_class) %>%
  autoplot(type = "heatmap")
simple_cnn_val_res %>%
  roc_curve(truth = label, .pred_1) %>%
  autoplot() +
  labs(
    title = "Receiver operator curve for Amazon Reviews"
  )
# Best CNN accuracy
cnn_best_acc <- max(simple_cnn_val_history$metrics$val_accuracy)

# Lowest CNN Loss
cnn_lowest_loss <- min(simple_cnn_val_history$metrics$val_loss)
results_table <- bind_rows(results_table,
                           tibble(Index = "3",
                                  Method = "CNN",
                                  Accuracy = cnn_best_acc,
                                  Loss = cnn_lowest_loss))

kable(results_table, "simple",caption="CNN Model Results\\label{tbl:cnn_results_table}")
# %>%
#     kable_styling(latex_options=c("HOLD_position"), font_size=7)
sep_cnn_model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words + 1, output_dim = 16,
                  input_length = max_length) %>%
  # layer_batch_normalization() %>%
  layer_dropout(0.2) %>%
  layer_separable_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
  layer_separable_conv_1d(filter = 32, kernel_size = 5, activation = "relu") %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_separable_conv_1d(filter = 64, kernel_size = 5, activation = "relu") %>%
  layer_separable_conv_1d(filter = 64, kernel_size = 5, activation = "relu") %>%
  layer_global_average_pooling_1d() %>%
  layer_dropout(0.2) %>%
  # layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

sep_cnn_model
# opt <- optimizer_sgd(lr = 0.001, decay = 1e-6)
# opt <- optimizer_adam()
# opt <- optimizer_sgd()
opt <- optimizer_adam(lr = 0.0001, decay = 1e-6)
sep_cnn_model %>% compile(
  optimizer = opt,
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

```
sep_cnn_val_history <- sep_cnn_model %>%
  fit(
    x = amazon_analysis,
    y = label_analysis,
    batch_size = 128,
    epochs = 20,
    initial_epoch = 0,
    validation_data = list(amazon_assess, label_assess),
    callbacks = list(callback_early_stopping(
        monitor='val_loss', patience=2)),
    verbose = 2
  )

sep_cnn_val_history
plot(sep_cnn_val_history)
sep_cnn_val_res <- keras_predict(sep_cnn_model, amazon_assess, label_assess)
# head(sep_cnn_val_res)
kable(head(sep_cnn_val_res), format="simple", caption="sepCNN Model Predictions using validation data\\l
m3 <- metrics(sep_cnn_val_res, label, .pred_class)
kable(m3, format="simple", caption="sepCNN Model Metrics using validation data\\label{tbl:sep_cnn_val_r
sep_cnn_val_res %>%
  conf_mat(label, .pred_class) %>%
  autoplot(type = "heatmap")
sep_cnn_val_res %>%
  roc_curve(truth = label, .pred_1) %>%
  autoplot() +
  labs(
    title = "Receiver operator curve for Amazon Reviews"
  )
# Best sepCNN accuracy
sep_cnn_best_acc <- max(sep_cnn_val_history$metrics$val_accuracy)

# Lowest sepCNN Loss
sep_cnn_lowest_loss <- min(sep_cnn_val_history$metrics$val_loss)
results_table <- bind_rows(results_table,
                           tibble(Index = "4",
                                  Method = "sepCNN",
                                  Accuracy = sep_cnn_best_acc,
                                  Loss = sep_cnn_lowest_loss))

kable(results_table, "simple",caption="sepCNN Model Results\\label{tbl:sep_cnn_results_table}")
# %>%
#     kable_styling(latex_options=c("HOLD_position"), font_size=7)
library(keras)
library(tfdatasets)
library(reticulate)
library(tidyverse)
library(lubridate)
library(tfhub)

# A dependency of the preprocessing for BERT inputs
# pip install -q -U tensorflow-text
import("tensorflow_text")
```

```r
# You will use the AdamW optimizer from tensorflow/models.
# pip install -q tf-models-official
# to create AdamW optimizer
o_nlp <- import("official.nlp")

Sys.setenv(TFHUB_CACHE_DIR="C:/Users/bijoor/.cache/tfhub_modules")
Sys.getenv("TFHUB_CACHE_DIR")

train_file_path <- file.path("amazon_review_polarity_csv/amazon_train.csv")

batch_size <- 32

train_dataset <- make_csv_dataset(
  train_file_path,
  field_delim = ",",
  batch_size = batch_size,
  column_names = list("label", "title", "text"),
  label_name = "label",
  select_columns = list("label", "text"),
  num_epochs = 1
)


train_dataset %>%
  reticulate::as_iterator() %>%
  reticulate::iter_next() #%>%
  # reticulate::py_to_r()

# ----------------------

val_file_path <- file.path("amazon_review_polarity_csv/amazon_val.csv")

val_dataset <- make_csv_dataset(
  val_file_path,
  field_delim = ",",
  batch_size = batch_size,
  column_names = list("label", "title", "text"),
  label_name = "label",
  select_columns = list("label", "text"),
  num_epochs = 1
)


val_dataset %>%
  reticulate::as_iterator() %>%
  reticulate::iter_next()

# ----------------------------------

test_file_path <- file.path("amazon_review_polarity_csv/amazon_test.csv")


test_dataset <- make_csv_dataset(
  test_file_path,
```

```
  field_delim = ",",
  batch_size = batch_size,
  column_names = list("label", "title", "text"),
  label_name = "label",
  select_columns = list("label", "text"),
  num_epochs = 1
)

test_dataset %>%
  reticulate::as_iterator() %>%
  reticulate::iter_next()

rm(amazon_orig_train, amazon_orig_test, amazon_train, amazon_val)
rm(ids_train, train_file_path, test_file_path, val_file_path)
bert_preprocess_model <- layer_hub(
  handle = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3", trainable = FALSE, name='prepro
)
bert_model <- layer_hub(
  handle = "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/2",
  trainable = TRUE, name='BERT_encoder'
)
input <- layer_input(shape=shape(), dtype="string", name='text')

output <- input %>%
  bert_preprocess_model() %>%
  bert_model %$%
  pooled_output %>%
  layer_dropout(0.1) %>%
  # layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid", name='classifier')

# summary(model)
model <- keras_model(input, output)
epochs = 5
steps_per_epoch <- 2e6
num_train_steps <- steps_per_epoch * epochs
num_warmup_steps <- as.integer(0.1*num_train_steps)

init_lr <- 3e-5
opt <- o_nlp$optimization$create_optimizer(init_lr=init_lr,
                                           num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                                 optimizer_type='adamw')

model %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = opt,
    metrics = "accuracy"
  )

summary(model)
# 10000 will take approx 40 mins per epoch on my gpu/mem etc
# 1000 will take approx 4 mins per epoch on my gpu/mem etc
```

```
tr_count <- 10000
take_tr <- 0.8 * tr_count
train_slice <- train_dataset  %>%
  dataset_shuffle_and_repeat(buffer_size = take_tr * batch_size) %>%
  dataset_take(take_tr)

take_val <- 0.2 * tr_count
val_slice <- val_dataset  %>%
  dataset_shuffle_and_repeat(buffer_size = take_val * batch_size) %>%
  dataset_take(take_val)
epochs <- 5
seed = 42

history <- model %>%
  fit(
    train_slice,
    epochs = epochs,
    validation_data = val_slice,
    initial_epoch = 0,
    verbose = 2
  )
plot(history)
model %>% evaluate(test_dataset)
test_slice <- test_dataset  %>%
  dataset_take(100)

model %>% evaluate(test_slice)
# Best BERT accuracy
bert_best_acc <-  max(history$metrics$val_accuracy)

# Lowest BERT Loss
bert_lowest_loss <- min(history$metrics$val_loss)
results_table <- bind_rows(results_table,
                           tibble(Index = "5",
                                  Method = "BERT",
                                  Accuracy = bert_best_acc,
                                  Loss = bert_lowest_loss))

kable(results_table, "simple",caption="BERT Model Results\\label{tbl:bert_results_table}")
# %>%
#     kable_styling(latex_options=c("HOLD_position"), font_size=7)
# Stop the clock
# proc.time() - ptm
Sys.time()
    knitr::knit_exit()
```

Terms like generate and some will also show up.

Alphabetical Index


```
knitr::knit_exit()
```