

DS 745: Project 3

Eli (Ilya) Bolotin

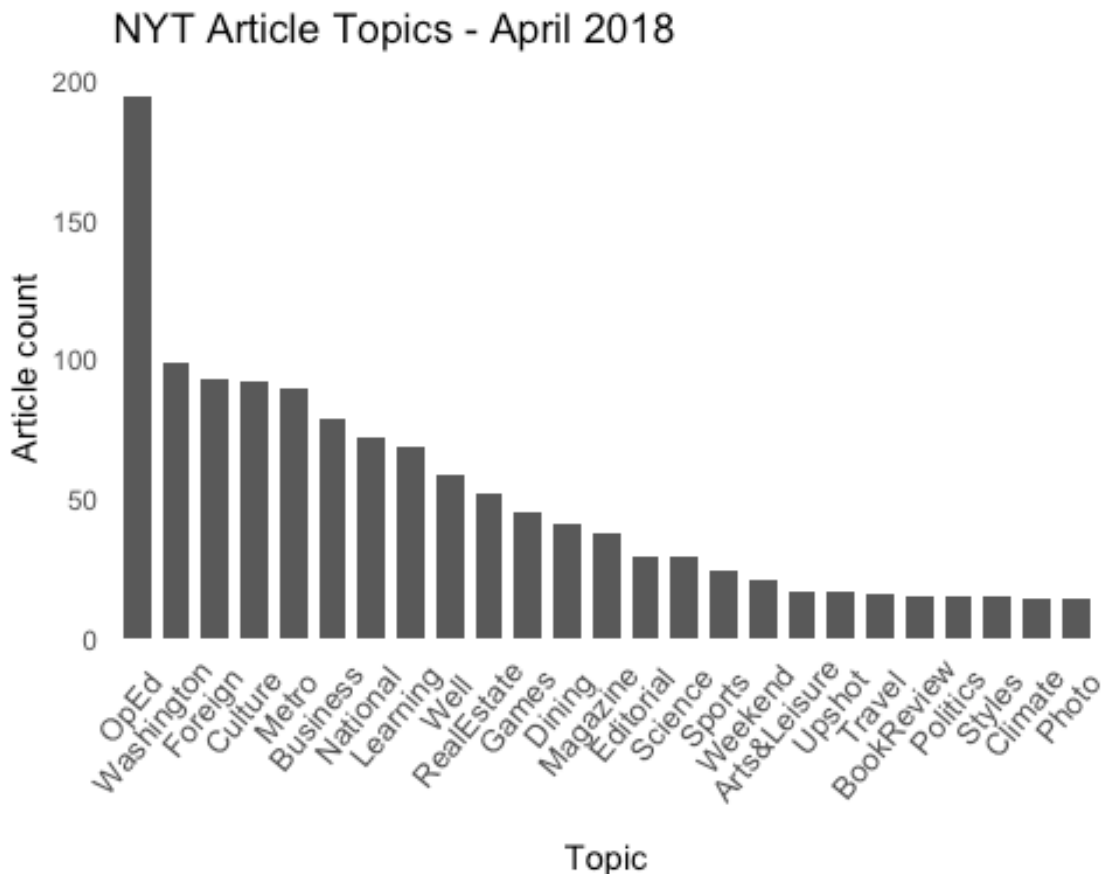
11/14/2019

Table of Contents

Dataset description	3
Objective	3
Methodology.....	3
Text visualization and analysis	3
Read in articles data and compute number of articles.....	3
Read in comments data.....	4
Preview comments and compute total classes.....	5
Split comments into train and test sets	5
Analyze text.....	5
Tokenize text.....	6
Examine tokenized data	6
Create binary vectors of words	7
Pad tokenized sequences.....	7
Create binary matrix of target variables for train/test sets	9
Create Keras model	9
Compile model.....	10
Define save/load directory	10
Train model.....	11
Plot model training.....	11
Load best model for plotting	12
Evaluate model performance	12
Findings along with the discussion	12
References.....	14

## 1	Former N.F.L. Cheerleaders' Settlement Offer: \$1 and a M...	Sports	Pro Footba...
## 2	E.P.A. to Unveil a New Rule. Its Effect: Less Science in...	Climate	Unknown
## 3	The New Noma, Explained	Dining	Unknown
## 4	How a Bag of Texas Dirt Became a Times Tradition	Insider	Unknown
## 5	Is School a Place for Self-Expression?	Learning	Unknown
## 6	Commuter Reprogramming	Magazine	Unknown

All 1324 articles above belong to 39 distinct topics. Below the top-25 most frequent article topics are plotted:



We will be training a neural network to predict the topics above based on **user comments** and **section heading/name**.

Read in comments data

```
# read in data
comments <- read.csv("nyt-comments/CommentsApril2018.csv", header = T)
comments <- as_tibble(comments[,c("commentBody", "newDesk", "sectionName")])

# examine dataset size
dim_comments <- dim(comments)
total_comments <- dim_comments[1]
```

There are a total of 264924 comments for April 2018.

Preview comments and compute total classes

Below is a small sample of user comments:

```
## # A tibble: 6 x 3
##   commentBody                                newDesk sectionName
##   <fct>                                <fct>    <fct>
## 1 "How could the league possibly refuse this offer? " Sports Pro Footba...
## 2 "So then the execs can be like \"yeah...we will sit down ... Sports Pro Footba...
## 3 I would not want to play chess against these cheerleaders... Sports Pro Footba...
## 4 "Could the cheerleaders join the Actors' Equity Associati... Sports Pro Footba...
## 5 Seeking conclusions which support preconceived ideas and ... Climate Unknown
## 6 "Pruitt: First eliminate the scientists and now eliminat... Climate Unknown
```

As mentioned above, there are 39 possible comment classes.

Next, we will remove any comments with an “Unknown” section name, as well as any comments with NA, and drop unused levels.

```
comments <- comments[comments$sectionName != 'Unknown',]
comments$commentBody <- as.character(comments$commentBody)
comments <- na.omit(comments)
comments <- droplevels.data.frame(comments)
num_rows <- nrow(comments)
```

We now have 117624 comments to train on.

Split comments into train and test sets

```
set.seed(39)
```

```
# shuffle data to ensure train and test subsets contain approximately normally distributed classes
```

```
new_indices <- sample(num_rows, replace = FALSE)
comments <- comments[new_indices,]
```

```
# sample train and test indices
```

```
num_train_samples <- round(num_rows * 0.90)
train_indices <- sample(1:num_rows, num_train_samples, replace = FALSE)
test_indices <- setdiff(1:num_rows, train_indices)
```

```
# create train/test datasets
```

```
x_train <- comments[train_indices, c("commentBody", "sectionName")]
y_train <- comments$newDesk[train_indices]
x_test <- comments[test_indices, c("commentBody", "sectionName")]
y_test <- comments$newDesk[test_indices]
```

Analyze text

The comments/text need to be converted into machine readable form (numeric form) before we can work with them. Let's first analyze how many unique words we have in our dataset. Note that

this is just a rough estimate, because we have not (yet) filtered symbols and other non-needed characters from the comments.

```
comment_id <- tibble(idx = length(x_train), text = as.character(x_train))
tokenized_tibble <- comment_id %>%
  unnest_tokens(word, text)
num_unique_words <- length(unique(tokenized_tibble$word))
```

There are approximately 74406 unique words in our corpus.

Tokenize text

Create a tokenizer that will map words to numbers for every comment (up to a maximum of unique words).

```
# create a tokenizer that maps words to numbers for every comment
vocab_len = num_unique_words
tokenizer <- text_tokenizer(num_words = vocab_len) %>%
  fit_text_tokenizer(x_train$commentBody)
```

Examine tokenized data

```
# Get actual number of unique words from tokenizer
vocab_len <- length(tokenizer$word_index)
```

```
# View first/last word indices
head(tokenizer$word_index)
```

```
## $the
```

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

```
##
```

```
## $to
```

```
## [1] 2
```

```
##
```

```
## $and
```

```
## [1] 3
```

```
##
```

```
## $of
```

```
## [1] 4
```

```
##
```

```
## $a
```

```
## [1] 5
```

```
##
```

```
## $br
```

```
## [1] 6
```

```
tail(tokenizer$word_index)
```

```
## $`american`
```

```
## [1] 82094
```

```
##
```

```
## $chainsaw
## [1] 82095
##
## $supercede
## [1] 82096
##
## $`'expertise'`
## [1] 82097
##
## $`'ghost'`
## [1] 82098
##
## $wheeled
## [1] 82099
```

Create binary vectors of words

One-hot-coding every word for every comment (i.e. producing a binary word-vector for every word of every comment) is very computationally expensive. Instead, with neural networks we can use an embedded layer to represent word-sequences in more compact form (by using a fixed number of coefficients).

To do this, we need to vectorize every comment according to the tokenized word indices created above.

```
# vectorize train and test data
train_vectors <- texts_to_sequences(tokenizer, x_train$commentBody)
test_vectors <- texts_to_sequences(tokenizer, x_test$commentBody)

# view sample sequence for first comment
head(train_vectors, 1)

## [[1]]
## [1] 366 5290 5215 93 827 22 2 70 1 3905 22 7387 3 69 436
## [16] 58 19 8400 9022 6 6 36 534 284 430 284 28 496 39 91
## [31] 48 284 13 72
```

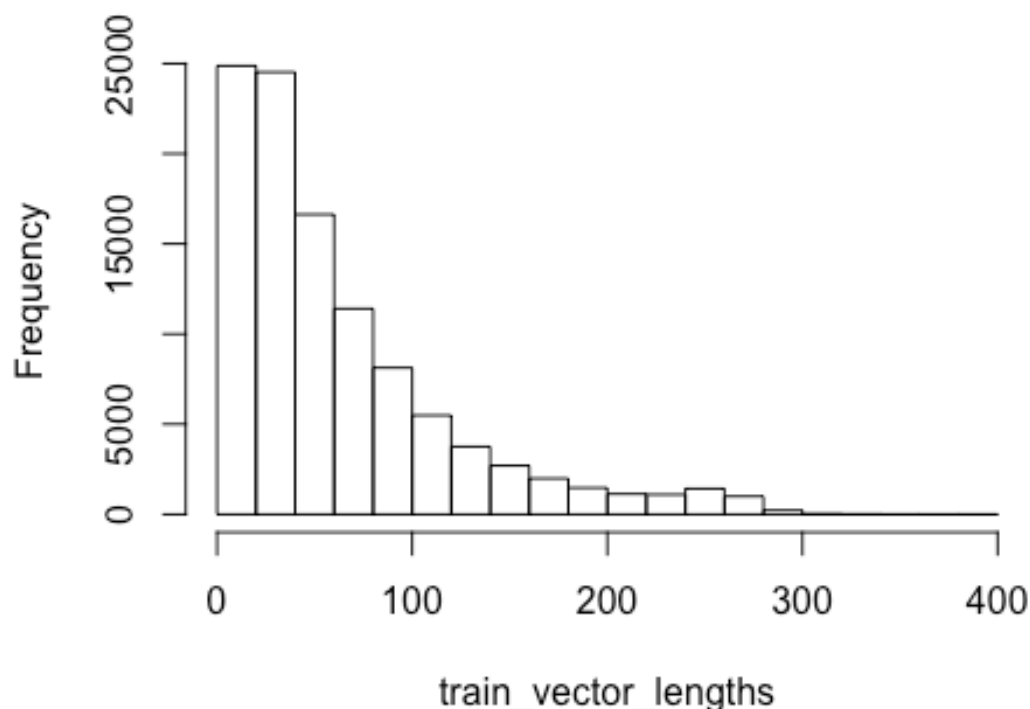
Pad tokenized sequences

Due to the fact that every word-sequence is of variable length, each sequence should be standardized to a fixed length before it can be fed to the neural network (which expects every input to have a common size).

First: Analyze the length of comments

```
train_vector_lengths <- sapply(train_vectors, length)
hist(train_vector_lengths, main = "Train Vector Lengths Distribution")
```

Train Vector Lengths Distribution



```
quantiles = quantile(train_vector_lengths, probs = c(0.05, .95))
```

The histogram shows a highly right-skewed distribution. We can see that length bottoms out near 300 words per comment (sequence). However, the 95% quantile is equal to 195 words per comment. So we can use this length as the input_size.

Next: Set a max sequence length to 195 words and pad vectors less than this with zeros.

```
# set input dimensions
vector_len = quantiles[2]
```

```
# Pad sequences
```

```
x_train <- pad_sequences(train_vectors, maxlen = vector_len, padding = "post")
x_test <- pad_sequences(test_vectors, maxlen = vector_len, padding = "post")
```

```
# View padded sequence (1 sequence shown)
```

```
head(x_train, 1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]  366 5290 5215   93  827   22    2   70    1 3905    22  7387    3    69
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]   436    58    19  8400  9022    6    6    36   534   284   430   284
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]    28   496    39    91    48   284    13    72    0    0    0    0
```



```
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116] [,117] [,118]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137] [,138]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,159] [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,169] [,170] [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,179] [,180] [,181] [,182] [,183] [,184] [,185] [,186] [,187] [,188]
## [1,]      0      0      0      0      0      0      0      0      0      0      0      0
##      [,189] [,190] [,191] [,192] [,193] [,194] [,195]
## [1,]      0      0      0      0      0      0      0
```

Above is an example of a padded sequence.

Create binary matrix of target variables for train/test sets

```
y_train <- as.factor(y_train)
y_train_numeric <- as.numeric(y_train)-1
y_train_bin <- to_categorical(y_train_numeric, num_classes)

y_test <- as.factor(y_test)
y_test_numeric <- as.numeric(y_test)-1
y_test_bin <- to_categorical(y_test_numeric, num_classes)
```

Create Keras model

Below is the neural network architecture. We are using a sequential model with 4 fully connected layers, 1 pooling layer, and 2 dropout layers. Notice the use of the embedding layer to represent vectorized words in compact form (input dim = 82099 and output = 300).

```
# create model
model <- keras_model_sequential()
```

```
# define layers
model %>%
  layer_embedding(input_dim = vocab_len, output_dim = 300) %>%
  layer_global_average_pooling_1d() %>%
  layer_dense(units = 150, activation = "relu") %>%
  layer_dropout(rate = 0.15) %>%
  layer_dense(units = 75, activation = "relu") %>%
  layer_dropout(rate = 0.15) %>%
  layer_dense(units = num_classes, activation = "softmax")
```

```
# view summary
summary(model)
```

```
## Model: "sequential"
```

```
##
```

## Layer (type)	Output Shape	Param #
## embedding (Embedding)	(None, None, 300)	24629700
## global_average_pooling1d (GlobalAve	(None, 300)	0
## dense (Dense)	(None, 150)	45150
## dropout (Dropout)	(None, 150)	0
## dense_1 (Dense)	(None, 75)	11325
## dropout_1 (Dropout)	(None, 75)	0
## dense_2 (Dense)	(None, 39)	2964

```
## =====
## Total params: 24,689,139
## Trainable params: 24,689,139
## Non-trainable params: 0
##
```

Compile model

Define gradient descent optimizer, loss function, and metrics

```
model %>% compile(
  optimizer = 'adam',
  loss = 'categorical_crossentropy',
  metrics = 'accuracy'
)
```

Define save/load directory

```
# create checkpoints folder to save model while training
checkpoint_dir <- "checkpoints"
```

```
dir.create(checkpoint_dir, showWarnings = FALSE)
filepath <- file.path(checkpoint_dir, "best_model.hdf5")
```

Train model

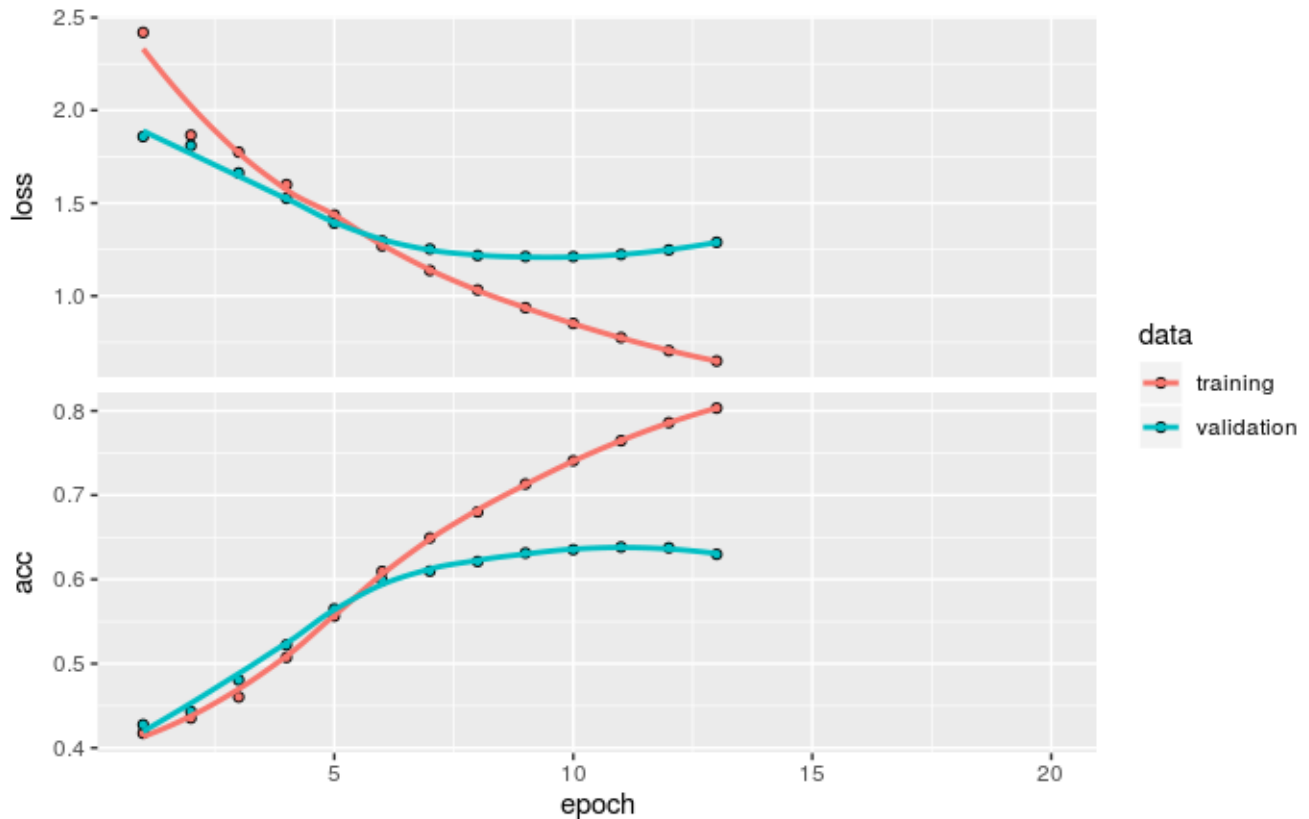
```
# Create checkpoint callback
cp_callback <- callback_model_checkpoint(
  filepath = filepath,
  save_weights_only = TRUE,
  save_best_only = TRUE,
  verbose = 0
)

early_stopping <- callback_early_stopping(patience = 3)

# train model
history <- model %>% fit(
  x_train, y_train_bin,
  epochs = 20,
  batch_size = 1280,
  validation_split = 0.20,
  verbose = 1,
  callbacks = list(cp_callback, early_stopping) # pass callback to training
)
```

Plot model training

```
plot(history)
```



Evaluate model performance

```
model %>% evaluate(x_test, y_test_bin, verbose = 0)
```

```
## $loss
## [1] 1.226128
##
## $accuracy
## [1] 0.6327956
```

Findings along with the discussion

As we see from the evaluation of the model on the test data, the model produces **~63%** classification accuracy on the test set (10599 observations and 39 classes). This is not a good result.

The training plot is highly indicative of what is happening here. From the training plot we see that model begins to overfit the training set by the 5th epoch. By the 10th epoch, validation accuracy has peaked at ~63%. (Training accuracy peaks at 81% at the ~13th epoch.) After the 13th epoch, the model suspends training because validation loss has not increased in 3 epochs (known as patience).

Why does the model have poor prediction accuracy on the test set? This is due to the generic nature of reader comments for NYT articles (and the topics created for them). Many comments are

simply not characteristic of any singular topic. Take for example this subset of comments with a *newsDesk* title of 'Sports' and *sectionName* of 'Pro Football':

```
slice(comments[comments$newDesk == "Sports" & comments$sectionName == "Pro
Football",], 50:60)

## # A tibble: 11 x 3
##   commentBody                                newDesk sectionName
##   <chr>                                <fct>    <fct>
## 1 "Good move, Cheerleaders !<br/>Poor NFL....they will hav... Sports    Pro Footba...
## 2 Their job is to be literal sex objects. Instead of askin... Sports    Pro Footba...
## 3 Are cheerleaders in the NFL worth keeping around? They m... Sports    Pro Footba...
## 4 "If the women cheerleaders/dancers find it demeaning to ... Sports    Pro Footba...
## 5 How many women tune in to NFL games to watch cheerleader... Sports    Pro Footba...
## 6 I thought the cheerleaders were there to enhance the mal... Sports    Pro Footba...
## 7 "If men had filed this lawsuit, citing harassment, low p... Sports    Pro Footba...
## 8 Why even have cheerleaders, they are not part of televis... Sports    Pro Footba...
## 9 NFL cheerleaders are underpaid, discriminated against an... Sports    Pro Footba...
## 10 "A savvy and excellent play by the women! "           Sports    Pro Footba...
## 11 Giving in to 'uppity women'                           Sports    Pro Footba...
```

The conversation in these comments is about harassment, lawsuits, and discrimination against cheerleaders. There is no actual conversation about sports, athletes, or NFL games. The more accurate classification topic here is *politics* or perhaps *culture*. Such is also the case with reader comments from *NYT April 2017* regarding Colin Kaepernick. When Kaepernick created a political movement by taking a knee during the national anthem of a football game, all of the comments in response to the subsequent NYT article had to do with the political repercussions of his actions (as well as the reactions of Donald Trump) – thus diluting any actual sports talk from comments labeled as “sports”. This is naturally confusing to neural nets.

Specifically in this project, the model (which assigns weights to words) has a difficult time correctly classifying statements in the validation set that consist of the same words (but in different context) found in the training set. In the training set the model updates its parameters to minimize the loss function, thus eventually achieving a perfect recognition of which *specific* comments (meaning – a specific sequence of words) belong to distinct topics. But when these same weights are applied to the validation set, which consists of largely the same words but in different context, the model cannot achieve similar accuracy – because inherently generic words in different order can belong to many topics. As such, the model cannot distinguish the correct classes with high enough probability. This is known as overfitting.

In conclusion, given the small amount of data used in this project and the short duration of training, the neural network cannot yet classify polysemantic comments beyond ~61% accuracy.

References

- [1] NYT Dataset, <https://www.kaggle.com/aashita/nyt-comments>
- [2] Keras for R, <https://blog.rstudio.com/2017/09/05/keras-for-r/>
- [3] Intro to text classification, <https://www.onceupondata.com/2019/01/21/keras-text-part1/>
- [4] Preparing data for NLP,
https://shirinsplayground.netlify.com/2019/01/text_classification_keras_data_prep/
- [5] Text classification in R, <https://www.jla-data.net/eng/vocabulary-based-text-classification/>