Neural Net Documentation

Sebastian Knigge

5 8 2019

1 Setup

Following packages were used in this script:

```
# loading packages
library(keras)
library(gutenbergr)
library(dplyr)
library(tensorflow)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(sampling)
```

2 Example 1 (6 books)

2.1 Get data

This is the essential step for setting up the neural net. These functions include the sampling procedure from the *gutenbergr* library

```
sampling_books <- function(seed=1234, n=20){
  # sample n books from the whole library
  set.seed(seed)
  gutenberg_works() %>%
    # select works with title
    dplyr::filter(!is.na(title)) %>%
    # set the sample sitze
    sample_n(n) %>%
    # set a special download link
    gutenberg_download(
      mirror = "http://mirrors.xmission.com/gutenberg/")
}

set_up_books <- function(n_books=4, seed=1992){
  # initial book sample
  books <- sampling_books(n=n_books, seed=seed)
  by_chapter <- books %>%
    group_by(gutenberg_id) %>%
    # split in chapters
    mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
    ungroup() %>%
    # exclude books without chapters
    dplyr::filter(chapter > 0)
```

```r
  return(by_chapter)
}

shorten_titles <- function(titles){
  # shorten very long book titles by setting
  # a subset of characters of the first line
  # of the title
  sub_inds <- titles %>%
    regexpr(pattern="\\n|\\r")-1
  sub_inds[sub_inds<0] <- nchar(titles)[sub_inds<0]
  titles %>%
    substr(1,sub_inds)
}

get_titles <- function(x, n_books){
  # get the sampled gutenberg_ids
  unique_ids <- x %>%
    select(gutenberg_id) %>%
    unique() %>% unlist()
  # get the titles
  titles <- gutenberg_works() %>%
    dplyr::filter(gutenberg_id %in% unique_ids) %>%
    select(gutenberg_id, title, author) %>%
    mutate(title=shorten_titles(title))
  # get the number of gutenberg ids
  len <- nrow(titles)
  if(n_books!=len) warning(paste("--- ",n_books-len,
                                 " books have 0 chapters --- "))
  # the output as a list
  ret <- list(
    titles=titles,
    len=len
  )
  return(ret)
}

append_by_chapter <- function(x=by_chapter, n_books, seed_index=1){
  # append the books matrix until
  # we get the desired number of books n_books
  titles <- get_titles(x, n_books)
  n <- titles$len
  while (n<n_books) {
    book2add <- sampling_books(n=1, seed=seed_index)
    by_chapter_add <- book2add %>%
      group_by(gutenberg_id) %>%
      # split in chapters
      mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
      ungroup() %>%
      # exclude books without chapters
      dplyr::filter(chapter > 2)
    titles2add <- get_titles(by_chapter_add, 1)
    # adding the book to by_chapter if there are chapters in the
    # book plus it is not in the data already
```

```r
    if (titles2add$len==1) if(!titles2add$titles$gutenberg_id%in%titles$titles$gutenberg_id) {
      x <- bind_rows(x, by_chapter_add)
    }
    n<-get_titles(x, n)$len
    seed_index <- seed_index+1
  }
  return(x)
}

exclude_stop_words <- function(x){
  # unite chapter and document title
  by_chapter_word <- x %>%
    unite(document, gutenberg_id, chapter) %>%
    # split into words
    unnest_tokens(word, text)
  # import tibble stop words
  data(stop_words)
  # find document-word counts
  word_counts <- by_chapter_word %>%
    # exclude stop words
    anti_join(stop_words) %>%
    # count each word by chapter
    count(document, word, sort = TRUE) %>%
    ungroup()
  return(word_counts)
}

convert_to_dtm <- function(x, minfq = 2){
  # get into a format lda can handle
  chapters_dtm <- x %>%
    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%
    # reduce by low frequencies
    dtm_remove_lowfreq(minfreq = minfq)
  return(chapters_dtm)
}


convert_to_dtm_2 <- function(x, n=n, minfq = 2, top=10000){
  # get into a format lda can handle
  chapters_dtm <- x %>%
    select(doc_id=document, term=word, freq=n) %>%
    document_term_matrix() %>%
    # reduce by low frequencies
    dtm_remove_tfidf(top=top)
  return(chapters_dtm)
}

# convert x matrix into a form such that it can be used for tensorflow
adjust_tensor_format <- function(x){
  x_chapters <- apply(x, 1, function(x) as.matrix(x)) %>% t()
  topics <- x %>% rownames() %>% as_tibble() %>%
    separate(value, c("gutenberg_id", "chapter"), sep = "_", convert = TRUE) %>%
```

```
    select(gutenberg_id) %>%
    # split joint name of book and chapter
    as.matrix %>% as.factor() %>% as.integer()
  # one hot encoding for the chapters (y)
  topics_categorical <- topics %>% -1 %>%
    to_categorical()
  ret <- list(
    x=x_chapters,
    y=topics_categorical,
    topics=topics
  )
  return(ret)
}
```

2.1.1 Sample corpus

Now we can use all these functions to get to the initial corpus sample. In this example 6 books are choosen.

```
n_books <- 6
by_chapter <- set_up_books(n_books=n_books, seed=222)
get_titles(by_chapter, n_books)
```

```
## Warning in get_titles(by_chapter, n_books): --- 1 books have 0 chapters ---
```

```
## $titles
## # A tibble: 5 x 3
##   gutenberg_id title                                   author
##          <int> <chr>                                   <chr>
## 1           11 Alice's Adventures in Wonderland        Carroll, Lewis
## 2         3096 Beatrice                                Haggard, H. Rider~
## 3        25603 Detailed Minutiae of Soldier life in the~ McCarthy, Carlton
## 4        47402 Along Alaska's Great River              Schwatka, Frederi~
## 5        49675 Hawkins Electrical Guide v. 5 (of 10)   Hawkins, N. (Nehe~
##
## $len
## [1] 5
```

The function set_up_books() returns a warning that one book seems to consist of only one chapter. In order to get a copus consisting out of 6 books, the function append_by_chaper() is used, which fills up the corpus to the desired number of books.

```
appended_by_chapter <- append_by_chapter(x=by_chapter, n_books = n_books)
word_counts <- exclude_stop_words(appended_by_chapter)
```

```
## Joining, by = "word"
```

In table 1 the sampled titles for the book sample with the seed 222 are displayed. It appears through the function append_by:chapter() one book was added, called “My Novel” — Volume 04“.

```
titles <- get_titles(appended_by_chapter, n_books)
titles$titles %>% stargazer(summary=FALSE, font.size = "footnotesize",
                            header=FALSE, title="Book-titles", rownames=FALSE,
                            label="titles:6books")
```

Table 1: Book-titles

gutenberg_id	title	author
11	Alice's Adventures in Wonderland	Carroll, Lewis
3096	Beatrice	Haggard, H. Rider (Henry Rider)
7705	"My Novel" — Volume 04	Lytton, Edward Bulwer Lytton, Baron
25603	Detailed Minutiae of Soldier life in the Army of Northern Virginia, 1861-1865	McCarthy, Carlton
47402	Along Alaska's Great River	Schwatka, Frederick
49675	Hawkins Electrical Guide v. 5 (of 10)	Hawkins, N. (Nehemiah)

2.1.2 Reduction of the dimensionality

In the set up we have another parameter to adjust. The function *convert_to_dtm* takes the parameter *minfq*, which is used to reduce the "bag of words" (i.e. dimensionality). *minfq* is the minimum frequency for the bag of words dictionary. I will refer to this as "embedding". Let us set it to 2 in this case, meaning that we include a word only if the frequency is 2 or more.

```
chapters_dtm <- convert_to_dtm(word_counts, minfq=2)
adjusted_format <- adjust_tensor_format((chapters_dtm))
ncol(chapters_dtm)
```

```
## [1] 10685
```

Let us compare it to the case if we include all words.

```
chapters_dtm_all <- convert_to_dtm(word_counts, minfq=0)
adjusted_format_all <- adjust_tensor_format((chapters_dtm))
ncol(chapters_dtm_all)
```

```
## [1] 17961
```

We also want to compare this to a reduction of the word dictionary by the tfidf. For the sake of comparison the reduction is made to the same value as used above via minfreq=2 (i.e. 10685 words).

```
chapters_dtm_tfidf <- convert_to_dtm_2(word_counts, top=10685)
adjusted_format_tfidf <- adjust_tensor_format((chapters_dtm))
ncol(chapters_dtm_tfidf)
```

```
## [1] 10685
```

2.1.3 Splitting and Fitting

The following function splits the sample in an manner, such that each cluster is eually to its size represented in the test data and the validation data.

```
sample_cluster_wise <- function(data, test_ratio=0.1, val_ratio=0.2, seed=1234){
  X <- data$x; y <- data$y
  cluster=data$topics
  set.seed(seed)
  {
  # setting the absolute number of observations for the sample of each cluster
  n_test <- (table(cluster)*test_ratio) %>% floor() %>%
    # use at least one observation of each cluster
    sapply(., function(x) max(x,1))
  n_val <- (table(cluster)*val_ratio) %>% floor() %>%
    # use at least one observation of each cluster
    sapply(., function(x) max(x,1))
```

```r
  # function to get the correct sample indices for validation and test sample
  samp_ind <- function(i, n_list) which(cluster==i) %>% sample(n_list[i])
  test_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_test)) %>%
    unlist()
  val_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_val)) %>%
    unlist()
  }
  ret <- list(partial_x_train=X[-c(val_indices,test_indices),],
              partial_y_train=y[-c(val_indices,test_indices),],
              x_val=X[val_indices,], y_val = y[val_indices,],

              x_test=X[test_indices,], y_test = y[test_indices,])
  return(ret)
}
```

The following two functions are setting up the model and evaluate the goodness of fit.

```r
# The whole model is set up and trained within this function
set_up_n_fit <- function(split, books_n=n_books){
  # starting with 64 neurons and scaling it down to 46 in the
  # mid layer turned out to be a well predicting model
  model <- keras_model_sequential() %>%
    layer_dense(units=64, activation="relu", input_shape=ncol(split$partial_x_train)) %>%
    layer_dense(units=46, activation="relu") %>%
    # we want to classify for as many categories as books
    layer_dense(units=books_n, activation="softmax")

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

  history <- model %>% fit(
    split$partial_x_train,
    split$partial_y_train,
    # from experience the model tends to
    # overfit for more than 5 epochs
    epochs=5,
    batch_size=512,
    validation_data=list(split$x_val,split$y_val)
  )
  return(
    list(history=history,
         model=model))
}


# making a prediction on the test data and calculating the
# misspecification rate; we also want to save the true categories and the predicted ones
evaluate_model <- function(model_fit, y=split$y_test, x=split$x_test) {
  prediction <- model_fit %>% predict(x)
  pred <- apply(prediction, 1, which.max)
```

```r
  true_value <- apply(y, 1, which.max)
  misspecified <- sum(!pred==true_value)/length(pred)
  ret <- list(misspecified=misspecified,
              # the function also dicloses the true and the predicted
              # values for exact evaluation, if needed
              pred=pred, true_value=true_value)
  return(ret)
}
```

Now we can start applying the model:

```r
tim1 <- Sys.time()
n <- 59
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format, seed=i*2)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
```

The results of the misclassification rate over 59 splits and fits of the model are:

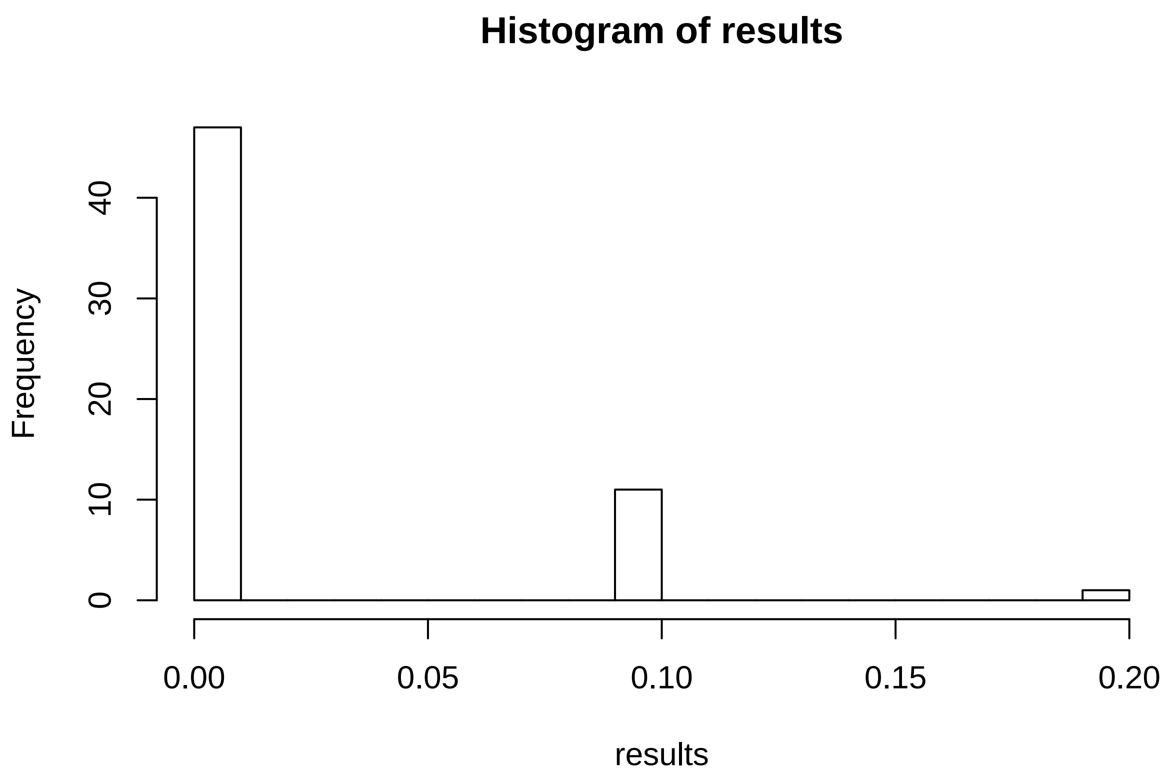
```r
results %>% summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.02203 0.00000 0.20000
```

```r
results %>% var
```

```
## [1] 0.002092344
```

```r
hist(results, breaks = 20)
```



... and the time:

```
# mfreq=2
(u1 <- tim2-tim1)
```

```
## Time difference of 3.646245 mins
```

We now try this procedure using the full bag of words:

```
tim1_all <- Sys.time()
n <- 59
results_all <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_all, seed=i*2)
  results_all[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2_all <- Sys.time()
```

The results of the error rate

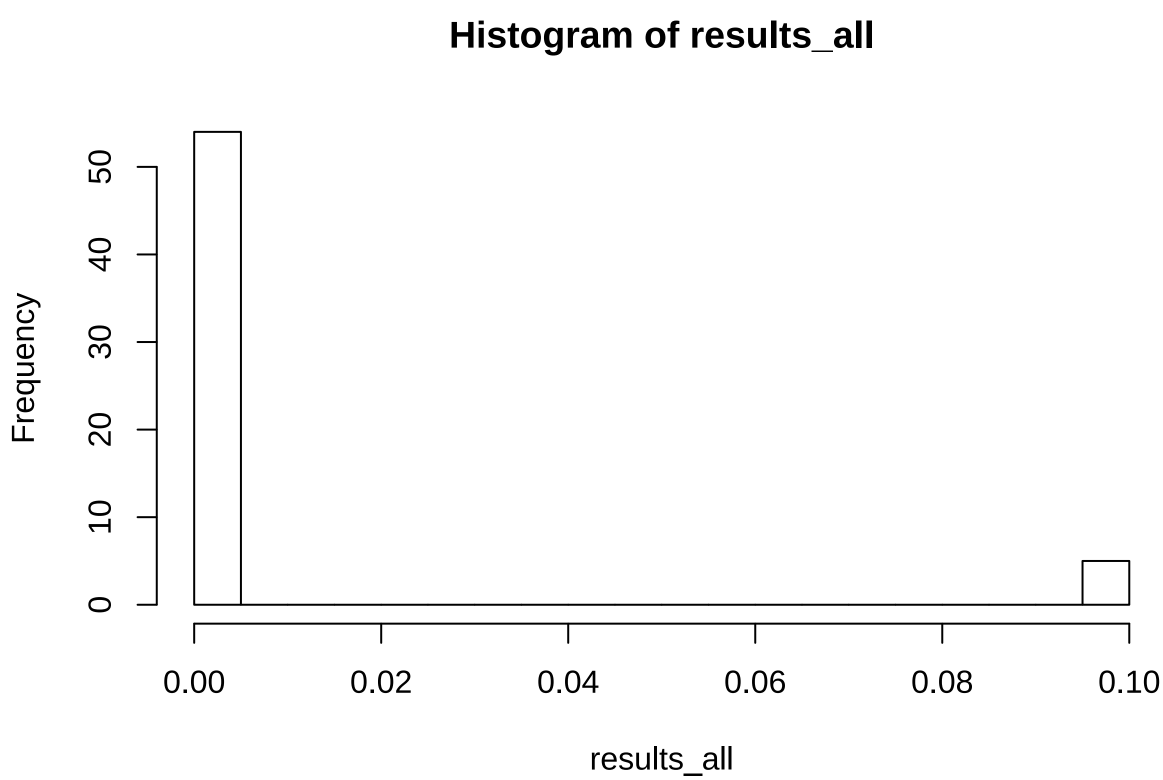
```
results_all %>% summary
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
## 0.000000 0.000000 0.000000 0.008475 0.000000 0.100000
```

```
results_all %>% var
```

```
## [1] 0.0007890123
```

```
hist(results_all, breaks = 20)
```



and the time:

```
# mfreq=2
(u2 <- tim2_all-(tim1_all))
```

```
## Time difference of 10.28905 mins
```

Now using the tfidf reduced bag of words:

```
tim1_tfidf <- Sys.time()
n <- 59
results_tfidf <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(adjusted_format_tfidf, seed=i*2)
  results_tfidf[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2_tfidf <- Sys.time()
```

The results of the error rate

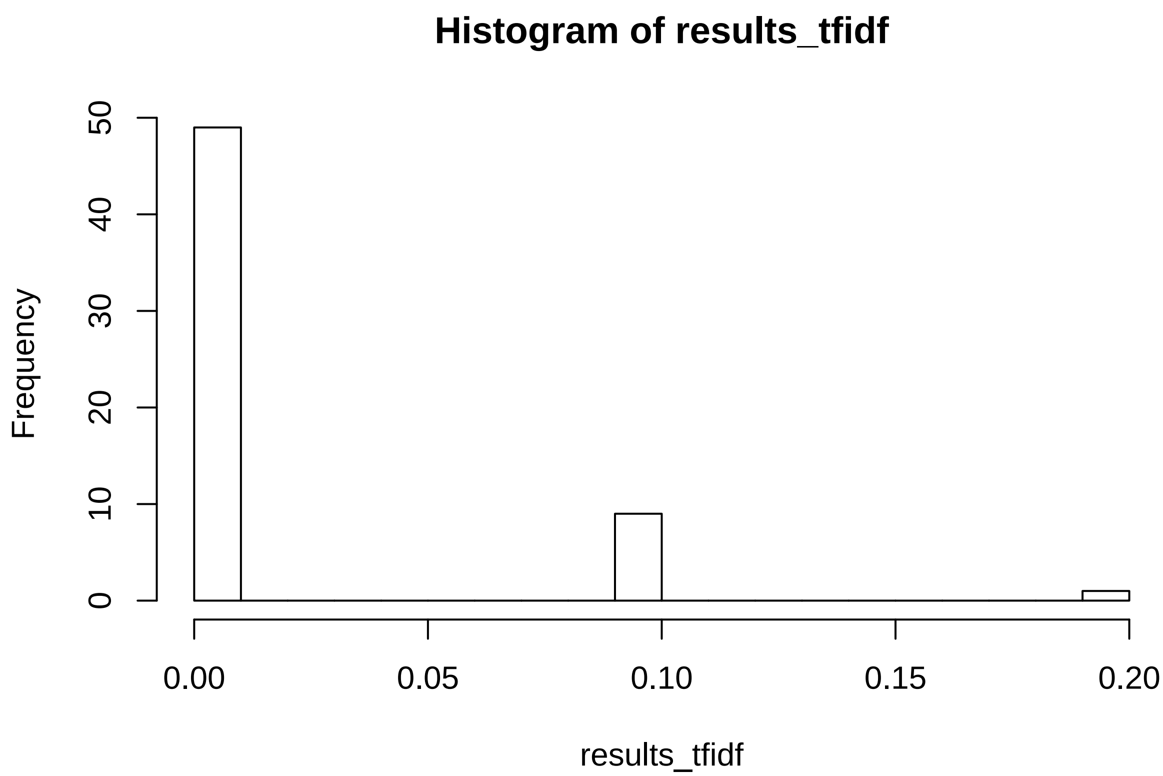
```
results_tfidf %>% summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.01864 0.00000 0.20000
```

```
results_tfidf %>% var
```

```
## [1] 0.001887785
```

```
hist(results_tfidf, breaks = 20)
```



and the time:

```
# mfreq=2
(u3 <- tim2_tfidf-(tim1_tfidf))
```

```
## Time difference of 17.29859 mins
```

Table 2 summarises the results.

```
performance_matrix <- data.frame(freq2.embedding=c(results%>%mean, u1),
           all.embedding=c(results_all%>%mean, u2),
           tfidf=c(results_tfidf%>%mean, u3))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F,
                                  title="Performance of embeddings",
                                  label="performancematrix")
```

Table 2: Performance of embeddings

	freq2.embedding	all.embedding	tfidf
missc. rate	0.022	0.008	0.019
time	3.646	10.289	17.299

The time for the embedding using more than frequency 2 is very short. Whereas the embedding via tfidf with the same dimensionality takes very long. It certainly makes sense that the missclassification rate is better for embedding with the full vocabulary than for the other two embeddings.