Classification Challenge Code

13343

24 May 2021

Introduction

Full code for the best submitted model is presented in `code.rmd` and `code.pdf`. Only code necessary for running the model is shown below with limited output. Details of the pre-processing steps and custom features added are shown and described in `writeup.rmd` and `writeup.pdf`, alongside other models explored for previous submissions.

Read in the data and tranform to document-feature-matrix

```
# read in the data from csv files
train_data <- read.csv("train.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test.csv", stringsAsFactors = FALSE)

# create two cprpuses from the comment text
train_corpus <- corpus(train_data, text_field = "text")
test_corpus <- corpus(test_data, text_field = "text")
```

```
## make train corpus dfm, no stemming
## (commented out are optional steps
## used on the other models)
mlp_train_dfm_nostem <- train_corpus %>%
  tokens(
    # remove_punct = TRUE,
    # remove_numbers = TRUE
    ) %>%
  ## remove stopwords
  tokens_remove(c(stopwords("en"))) %>%
  ## select features of at least 2 letters
  # tokens_select(min_nchar = 2L) %>%
  ## select bigrams
  # tokens_ngrams(n = 1:2) %>%
  dfm(
    # remove_symbols = TRUE,
    remove_url = TRUE,
    stem = FALSE,
    verbose = FALSE
    )
  ## trim to most frequent features
  # dfm_trim(min_docfreq = 10)
```

```
# stem the dfm
mlp_train_dfm_stem <- dfm(mlp_train_dfm_nostem,
                          stem = TRUE)
```

```
# perform the same steps on test dataset
mlp_test_dfm_nostem <- test_corpus %>%
  tokens() %>%
  tokens_remove(c(stopwords("en"))) %>%
  dfm(
    remove_url = TRUE,
    stem = FALSE,
    verbose = FALSE
    )

mlp_test_dfm_stem <- dfm(mlp_test_dfm_nostem,
                         stem = TRUE)
```

```
# find common words between datasets
mlp_common_feats <- intersect(featnames(mlp_train_dfm_stem),
                              featnames(mlp_test_dfm_stem))

# combine the train and test dfms on
# the common features
mlp_comb_dfm <- rbind(mlp_train_dfm_stem[,mlp_common_feats],
                      mlp_test_dfm_stem[,mlp_common_feats])
```

Add profanity count as custom feature

```
# read in profanity list from:
# https://www.cs.cmu.edu/~biglou/resources/bad-words.txt
cs_bad_words <- read.delim("bad-words.txt")

# load profanity lists from 'lexicon' package
profanity_list <- c(profanity_alvarez,
                    profanity_arr_bad,
                    profanity_banned,
                    profanity_racist,
                    as.vector(cs_bad_words))

# subset to only unique profanity words
profanity_list_unique <- unique(profanity_list)

# number of profanity words
length(profanity_list_unique)
```

```
## [1] 1222
```

```
# create dictionary from word list
profanity_dict <- dictionary(list(profanity_word = profanity_list_unique))
```

```r
# apply dictionary to dfm to get counts of profanity
mlp_train_profanity_dfm <- dfm_lookup(mlp_train_dfm_nostem,
                                      dictionary = profanity_dict)

mlp_test_profanity_dfm <- dfm_lookup(mlp_test_dfm_nostem,
                                     dictionary = profanity_dict)

# inspect one dfm to see feature added
mlp_train_profanity_dfm
```

```
## Document-feature matrix of: 15,000 documents, 1 feature (67.2% sparse) and 2 docvars.
##        features
## docs    profanity_word
##   text1              0
##   text2              0
##   text3              2
##   text4              0
##   text5              3
##   text6              2
## [ reached max_ndoc ... 14,994 more documents ]
```

Add sentiment and emotion as custom features

```r
# create copy of NRC dictionary to re-label emotions/sentiment
data_dictionary_NRC_copy <- data_dictionary_NRC

# save names for output
output_names <- names(data_dictionary_NRC)

# rename sentiment so that it doesn't overlap
# with existing features when combining dfm
names(data_dictionary_NRC_copy) <- c("anger_sent", "anticipation_sent",
                                     "disgust_sent", "fear_sent", "joy_sent",
                                     "negative_sent", "positive_sent", "sadness_sent",
                                     "surprise_sent", "trust_sent")

# apply dictionary to weighted
# training and testing data
train_sent_dfm <- dfm_lookup(dfm_weight(mlp_train_dfm_nostem,
                                        scheme = "prop"),
                             dictionary = data_dictionary_NRC_copy)

test_sent_dfm <- dfm_lookup(dfm_weight(mlp_test_dfm_nostem,
                                       scheme = "prop"),
                            dictionary = data_dictionary_NRC_copy)

# remove the insignificant emotions
# (no difference between anticipation and joy)
train_sent_dfm <- train_sent_dfm[, -c(2, 5)]
test_sent_dfm <- test_sent_dfm[, -c(2, 5)]
```

Add number of all caps words as custom feature

```
# remove URLS before counting capital letters
# used regex pattern from: https://stackoverflow.com/questions/2
# 6496538/extract-urls-with-regex-into-a-new-data-frame-column/26498790
url_pattern <- "http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|
[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+"

# replace urls for both corpora
train_corpus_no_url <- gsub(url_pattern, "", train_corpus)
test_corpus_no_url <- gsub(url_pattern, "", test_corpus)

# count number of capital words
# used regex pattern from: https://stackoverflow.com/questions/
# 33197733/how-to-count-capslock-in-string-using-r
train_num_capital_words <- str_count(train_corpus_no_url, "\\b[A-Z]{2,}\\b")
test_num_capital_words <- str_count(test_corpus_no_url, "\\b[A-Z]{2,}\\b")
```

Create final dfm

```
# add the custom features
mlp_comb_dfm_custom <- cbind(mlp_comb_dfm,
                             # add sentiment and emotion proportion as featurs
                             rbind(train_sent_dfm, test_sent_dfm),
                             # add profanity count as features
                             rbind(mlp_train_profanity_dfm, mlp_test_profanity_dfm))

final_dfm <- cbind(mlp_comb_dfm_custom,
                   # add the capital words count as a feature
                   rbind(as.dfm(data.frame(num_capital_words = train_num_capital_words)),
                         as.dfm(data.frame(num_capital_words = test_num_capital_words))))

# add back attack and doc id docvars
docvars(final_dfm, "attack") <- as.factor(c(docvars(mlp_train_dfm_stem, "attack"),
                                            # NAs for testing set
                                            rep(NA, 100000)))

docvars(final_dfm, "id") <- c(docvars(mlp_train_dfm_stem, "id"),
                              docvars(mlp_test_dfm_stem, "id"))

# inspect final dfm
final_dfm
```

```
## Document-feature matrix of: 115,000 documents, 26,382 features (99.9% sparse) and 2 docvars.
##        features
## docs    may contain detail regardless  , point will see inform add
##   text1   1       1      1          0  0     0    0   0      0   0
##   text2   0       0      0          1  2     1    1   1      1   1
##   text3   0       0      0          0  2     0    0   0      0   0
##   text4   0       0      0          0  1     0    0   0      0   0
```

```
##   text5   0       0      0          0  2     0    0   0      1   0
##   text6   2       0      0          0 27     0    0   3      0   0
## [ reached max_ndoc ... 114,994 more documents, reached max_nfeat ... 26,372 more features ]
```

Subset training, validation, and testing sets

```
# store indices for each dataset
tr <- c(1:12000)
val <- c(12001:15000)
te <- c(15001:115000)

# subset unweighted dfm
train_set <- final_dfm[tr,]
val_set <- final_dfm[val,]
test_set <- final_dfm[te,]

# weight the dfm using tf-idf
final_dfm_wt <- dfm_tfidf(final_dfm)

# subset weighted dfm
train_set_wt <- final_dfm_wt[tr,]
val_set_wt <- final_dfm_wt[val,]
test_set_wt <- final_dfm_wt[te,]
```

```
# subset the class labels as factors
train_class <- as.factor(train_data$attack[tr])
val_class <- as.factor(train_data$attack[val])

# inspect classes
summary(train_class)
```

```
##    0    1
## 9380 2620
```

```
summary(val_class)
```

```
##    0    1
## 2398  602
```

Multi-Layer Perceptron Model

Code for searching over different hyperparameters and tuning the model the best model is in the `writeup.Rmd` file and described in `writeup.pdf`.

```
# run the best mlp model
best_mod_mlp <- textmodel_mlp(
                  x              = train_set_wt[,],
                  y              = train_class,
                  units          = 512,
                  epochs         = 70,
```

```
                  optimizer      = "sgd",
                  loss           = "binary_crossentropy",
                  metrics        = "accuracy",
                  dropout        = 0.8,
                  class_weight   = list("0" = 0.2148, "1" = 0.7852),
                  verbose        = 1
                  )
```

```
# predict training and validation performance
mlp_tr_preds <- predict(best_mod_mlp, train_set_wt[,], type = "class")
mlp_val_preds <- predict(best_mod_mlp, val_set_wt[,], type = "class")

# create confusion matrices
mlp_tr_confmat <- table(mlp_tr_preds, train_class)
mlp_val_confmat <- table(mlp_val_preds, val_class)
```

```
# get performances
mlp_tr_perform <- confusionMatrix(mlp_tr_confmat, positive = "1", mode = "prec_recall")
mlp_val_perform <- confusionMatrix(mlp_val_confmat, positive = "1", mode = "prec_recall")

# display results in data frame
results_df <- data.frame(dataset   = c("training", "validation"),
                         precison  = c(round(mlp_tr_perform$byClass[5], 3),
                                       round(mlp_val_perform$byClass[5], 3)),
                         recall    = c(round(mlp_tr_perform$byClass[6], 3),
                                       round(mlp_val_perform$byClass[6], 3)),
                         f1        = c(round(mlp_tr_perform$byClass[7], 3),
                                       round(mlp_val_perform$byClass[7], 3)),
                         accuracy  = c(round(mlp_tr_perform$overall[1], 3),
                                       round(mlp_val$overall[1], 3))
                         ) %>%
  kbl(booktabs = T,
      row.names = F,
      caption = "MLP model performance with 70 epochs") %>%
  kable_styling(latex_options =  c("striped", "HOLD_position"),
                full_width = FALSE)
```

Table 1: MLP model performance with 70 epochs

dataset	precision	recall	f1	accuracy
training	0.8749	0.9553	0.9133	0.9604
validation	0.8720	0.6719	0.6893	0.8720

Predict labels for the test set and submit to competition

```
# predict the labels for the test set
# (had to split in to because of memory limit)
test_preds1 <- predict(best_mod_mlp,  test_set_wt[1:25000,], type = "class")
test_preds2 <- predict(best_mod_mlp,  test_set_wt[25001:50000,], type = "class")
```

```r
# unname the vectors
test_preds1 <- unname(test_preds1)
test_preds2 <- unname(test_preds2)

# combine the predictions
test_preds <- append(as.character(test_preds1),
                     as.character(test_preds2),
                     after = length(test_preds1))
```

```r
# combine predictions with ids
submission_df  <- data.frame(id = test_data$id,
                             attack = as.factor(test_preds))

# remove rownames and save to csv to submit
rownames(submission_df) <- NULL
write.csv(submission_df, "submission.csv", row.names = FALSE)
```

```r
# inspect predictions
table(as.factor(submission_df$attack)) %>%
  data.frame() %>%
  kbl(booktabs = T,
      col.names = c("Attack Label", "Frequency"),
      caption = "Model predictions for test data") %>%
  kable_styling(latex_options =  c("striped", "HOLD_position"),
                full_width = FALSE)
```

Table 2: Model predictions for test data

Attack Label	Frequency
0	77441
1	22559