Tweet_Prediction

John Cooper

4/2/2021

Instructions

This is a summary output of the code in R Markdown which was also submitted To Run the included RMD File, R studio is required. Rstudio Cloud is not recommended (it's slow) the desktop version is much faster. I could replicate this in python but I have much more experience in R, and my python skills are still a little amateur (working hard to remedy). To load the data, place the raw csv file in your working directory and run all code chunks and optionally knit to pdf (this will replicate this document)

Summary:

I have extensive training in several aspects of NLP including prediction which really came in handy with this. I was suspicious of over-fitting but after comparing results from training & testing, then shuffling the data and re-sampling, I'm still getting pretty much the same results. I'm confident that my model will perform well on the non-labeled data.

-
- Both Test and Training data results show a 99% accuracy, consistent outcomes are a good sign.
- Time to complete is under 60 seconds. (Installing packages may extend this time)
- Advanced pre processing (key to great outcome)
- LSA
- K-Fold Cross Validation
- Tokenize
- Clean and Truncate text
- TFIDF
- svmLinear (othere tested, this was the best)
- Large (30%) sample data used to validate results

Sources:

I have extensive training in NLP which i relied on heavily. I also used information from the following. Certain chunks of code I developed previously for a personal NLP project, this made development a little easier.

https://dataaspirant.com/support-vector-machine-classifier-implementation-r-caret-package/ https://seantrott.github.io/binary_classification_R/ https://stackoverflow.com/questions/4090169/elegant-way-to-check-for-missing-packages-and-install-them https://cran.r-project.org/web/packages/caret/caret.pdf https://cran.r-project.org/web/packages/quanteda/quanteda.pdf

Install and Load Packages if required

```
# Time the code execution
start.time <- Sys.time()

# if required, install and load each package
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
if (!require('e1071')) install.packages('e1071'); library('e1071')
if (!require('caret')) install.packages('caret'); library('caret')
if (!require('quanteda')) install.packages('quanteda'); library('quanteda')
if (!require('irlba')) install.packages('irlba'); library('irlba')
if (!require('randomForest')) install.packages('randomForest'); library('randomForest')
if (!require('doSNOW')) install.packages('doSNOW'); library('doSNOW')
if (!require('kernlab')) install.packages('kernlab'); library('kernlab')
```

Load data from Working Directory

```
# after manually placing the data in your working directory, Load the CSV
bos_b_twts_raw <- read_csv("boston_bombing_tweets.csv") #if error, add data to WD
```

Turn label data into categorical values

```
# turn the binary responses into a categorical value (factor)
bos_b_twts_raw$label <- as.factor(bos_b_twts_raw$label)
```

Filter for labeled data, split into train/test data (70/30)

```
# Filter for labeled data only
bos_b_twts_filt <- bos_b_twts_raw %>% filter(label != "NA")

# set seed for a consistent output
set.seed(12345)

# split labeled data into testing and training (70/30)
bos_b_twts_record <- createDataPartition(bos_b_twts_filt$label, times = 1,
                                         p = 0.7, list = FALSE)
```

```
bos_b_twts_train <- bos_b_twts_filt[bos_b_twts_record,]
bos_b_twts_test <- bos_b_twts_filt[-bos_b_twts_record,]
```

Tokenize, clean, stem, set text frequency requirements and prep LSA and SVD

Terms: dfm: Document Feature Matrix tfidf: Term frequency - inverse document frequency: This is a measure used to identify which words are important. lsa: latent Samentic ANalysis: svd:

```
# Use quanteda to preprocess the text into tokens
bos_b_twts_train_token2 <- tokens(bos_b_twts_train$text,
                                  what = "word",
                                  remove_numbers = TRUE,
                                  remove_punct = TRUE,
                                  remove_symbols = TRUE,
                                  split_hyphens = TRUE) %>%
  tokens_tolower()

# Further clean the text by removing stop words and stemming words
# Only use words with a frequency of 1.5% across 2+ tweets
bos_b_twts_train_dfm<-bos_b_twts_train_token2 %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  dfm() %>%
  dfm_trim( min_termfreq = round(nrow(bos_b_twts_train)*.02, 0),
            min_docfreq = 2) %>%
  dfm_tfidf()

#train_lsa
# Latent Semantic Analysis - for dimension reduction (very important step)
train_lsa <- irlba(t(bos_b_twts_train_dfm),
                   nv = ncol(bos_b_twts_train_dfm)-1, maxit = 300) # this creates
# a trunkated file  aprox 50 x 300
train_svd <- data.frame(Label = bos_b_twts_train$label,
                        ReviewLength= nchar(bos_b_twts_train$text),
                        train_lsa$v)
```

Lets preview what we have created

```
head(train_svd)
```

```
##   Label ReviewLength          X1           X2           X3           X4
## 1     1          146 -0.03765649 -0.025071096  0.002159484 -0.001179281
## 2     1          102 -0.03568436 -0.031620889 -0.065369507 -0.004789848
## 3     1          154 -0.01297031 -0.007927574  0.009917403 -0.014692882
## 4     0          104 -0.01454266  0.026444037 -0.004209206  0.001038199
## 5     1           74 -0.02576942 -0.008048393 -0.010047849  0.032153765
## 6     1           60 -0.02255234 -0.012203324  0.013208465 -0.041144738
##             X5           X6           X7            X8           X9
## 1 -0.045674212  0.034717382 -0.015623359 -0.0091058980 -0.004611227
## 2  0.023688179  0.004402869 -0.003508276 -0.0042222147 -0.013431551
## 3 -0.012505246 -0.016138537  0.011663457  0.0430660867  0.003534799
## 4 -0.001149614  0.005310636  0.021443075 -0.0040172678  0.009806726
## 5 -0.034431260 -0.018514693  0.003004893  0.0007836796 -0.040321573
## 6 -0.019123449 -0.001591594  0.003744319 -0.0094789057 -0.007961993
##            X10           X11          X12          X13           X14
## 1 -0.053754895 -8.434914e-02 -0.060327345  0.024750591 -9.936371e-03
## 2 -0.007146840  3.545834e-03  0.010254461 -0.008169350 -1.060689e-02
## 3  0.024279536  4.826950e-02  0.014275672  0.005997223 -1.599748e-02
## 4 -0.004740079  6.031493e-05 -0.012524968 -0.005457472 -2.495475e-05
## 5 -0.001589240 -1.154883e-02 -0.007728714 -0.020421565 -1.391598e-02
## 6 -0.006065475 -1.401973e-02  0.002896993 -0.003120249  1.809068e-03
##             X15          X16           X17          X18          X19
## 1  0.0189383970  0.025248862 -0.1255556793  0.042490671  0.119062400
## 2  0.0058722649 -0.001744755 -0.0011309471 -0.010527398 -0.002798773
## 3  0.0144822497 -0.001062444 -0.0226366107  0.021937820  0.017290141
## 4  0.0323004123 -0.009534472 -0.0005671778 -0.001481073 -0.009848740
## 5 -0.0003338022  0.000575828  0.0081587352 -0.008549121 -0.012852896
## 6 -0.0059200794 -0.011249230  0.0168765447 -0.012201533 -0.013853098
##            X20           X21          X22          X23          X24
## 1 -0.061958409  0.0005277255  0.035215863 -0.022386484  0.037788907
## 2 -0.028226187  0.0065537081  0.020144110  0.009284945  0.012024155
## 3  0.004475907 -0.0045197184 -0.014972255 -0.016397950  0.014683698
## 4 -0.004008576  0.0199018191  0.008269680 -0.004503323 -0.006785115
## 5 -0.022574492  0.0143725355  0.021618684 -0.003984582  0.015476013
## 6  0.015995132  0.0111261474 -0.002365813  0.008144364 -0.008013768
##             X25           X26          X27          X28         X29
## 1  0.0138949212  0.0073804126 -0.033778787  0.035152325 -0.01362456
## 2 -0.0014848023  0.0181365048 -0.011998485  0.007033284  0.01486295
## 3 -0.0038738344 -0.0131410753  0.008804965  0.003502548  0.01246531
## 4 -0.0085676786  0.0001569372  0.011628689  0.004960476  0.03431435
## 5  0.0002678741  0.0125111947 -0.019757223  0.001342817  0.02193979
## 6  0.0017753406  0.0161198518 -0.005587346 -0.036809942 -0.01514581
##            X30           X31          X32
## 1  0.010993602 -8.542018e-03 -0.013203782
## 2  0.036473748  1.192082e-02  0.018847473
## 3  0.004423949  4.852904e-05 -0.005510932
## 4 -0.012433839 -6.100376e-02  0.024867321
## 5  0.034142644 -5.301262e-03  0.035991923
## 6 -0.101096293  2.979707e-02  0.026710897
```

Prep the K-Fold cross validation

```r
# K Fold Cross Validation using the package caret
# Create folds for 10-fold cross validation
set.seed(48743)
cv.folds <- createMultiFolds(train_svd$Label, k = 10, times = 2)

# basically this will create 20 random samples
cv.cntrl <- trainControl(method = "repeatedcv", number = 10,
                         repeats = 2, index = cv.folds)
```

Speed up processing, This reduces run time from 30+ minutes to 2 minutes.

Note: this does not benefit if ran on Rstudio Cloud. Only speeds up processing if ran from desktop version of Rstudio, or on a VM with multiple cores.

```r
# Speed up processing in order to scale
# Create a cluster to work on 3 logical cores
# essentially 3 instances of Rstudio for processing
cl <- makeCluster(3, type = "SOCK")

# register the instance for faster processing
registerDoSNOW(cl)
```

Train the svmRadial Model using the previously defined Training data, then end the cluster

```r
# set seed for consistency
set.seed(12345)

# Train the Training Data
SVM_RB <- train(
  Label ~., data = train_svd, method = "svmLinear",
  trControl = cv.cntrl,
  preProcess = c("center","scale"))

# Processing is done, stop cluster.
on.exit(stopCluster(cl))
```

Review the prediction results based on training data.

This output will be compared to the test data output. A large discrepancy would suggest overfilling, or issues relating to the samples taken.

```r
# Review Results from Training data
preds <- predict(SVM_RB, train_svd)
confusionMatrix(preds, train_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1396    5
##          1    4 1395
##
##                Accuracy : 0.9968
##                  95% CI : (0.9939, 0.9985)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                   Kappa : 0.9936
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9971
##             Specificity : 0.9964
##          Pos Pred Value : 0.9964
##          Neg Pred Value : 0.9971
##              Prevalence : 0.5000
##          Detection Rate : 0.4986
##    Detection Prevalence : 0.5004
##       Balanced Accuracy : 0.9968
##
##        'Positive' Class : 0
##
```

Above are fantastic results, proceed cautiously, watch for signs of over fitting.

Apply the Model to the Testing Data

30% of the labeled data was with held, Now I'm going to use that data to test the model for accuracy.

Tokenize and clean the testing data as we did with the training data

```r
bos_b_twts_test_token <- tokens(bos_b_twts_test$text, what = "word",
                               remove_numbers = TRUE, remove_punct = TRUE,
                               remove_symbols = TRUE, split_hyphens = TRUE) %>%
  tokens_tolower()

# Further clean the text by removing stop words and stemming
# Only use words with a frequency of 2.5% across 2+ tweets
bos_b_twts_test_dfm<-bos_b_twts_test_token %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  dfm() %>%
  dfm_trim( min_termfreq = round(nrow(bos_b_twts_test)*.02, 0),
           min_docfreq = 2) %>%
  dfm_tfidf()
```

Match data formats (testing vs training)

```r
# Use quanteda to make sure training and testing data have exact same format
bos_b_twts_test_dfm <- dfm_select(bos_b_twts_test_dfm,
                                  pattern = bos_b_twts_train_dfm,
                                  selection = "keep")
bos_b_twts_test_matrix <- as.matrix(bos_b_twts_test_dfm)
```

Prep the data further

```
# taking the transpose of the singular matrix
sigma.inverse <- 1 / train_lsa$d

# transpose of the term matrix
u.transpose <- t(train_lsa$u)
test_svd <- t(sigma.inverse * u.transpose %*% t(bos_b_twts_test_dfm))
test_svd<-as.matrix(test_svd)

# Build the test data frame to feed into our trained machine learning model
# for predictions.
test_svd <- data.frame(Label = bos_b_twts_test$label,
                       ReviewLength= nchar(bos_b_twts_test$text),
                       test_svd)
```

Make predictions and output a confusion matrix and measures to assess the quality of the models performance on the testing data

```
# Make predictions on the test data set using our trained SVM.
preds <- predict(SVM_RB, test_svd)
confusionMatrix(preds, test_svd$Label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 600   6
##          1   0 594
##
##                Accuracy : 0.995
##                  95% CI : (0.9891, 0.9982)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.99
##
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
##             Specificity : 0.9900
##          Pos Pred Value : 0.9901
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5050
##       Balanced Accuracy : 0.9950
##
##        'Positive' Class : 0
##
```

Consistent accuracy across training and test indicates the model is performing well and is not over-fit. Also, experimented with different sample sizes and sampling methods which all produced similar results. I am comfortable that the model is not over fit.

Apply the model to the entire data set in preparation for submission.

the original data source was saved as raw, we will now prep that data as we

already have with the training and testing data

and applying our predictions

```r
# the original data source was saved as raw, we will now prep that data
# and applying our predictions
bos_b_twts_apply_token <- tokens(bos_b_twts_raw$text, what = "word",
                                remove_numbers = TRUE,
                                remove_punct = TRUE,
                                remove_symbols = TRUE,
                                split_hyphens = TRUE) %>%
  tokens_tolower()

# Further clean the text by removing stop words and stemming
# Only use words with a frequency of 2.5% across 2+ tweets
bos_b_twts_apply_dfm <- bos_b_twts_apply_token %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  dfm() %>%
  dfm_trim(min_termfreq = round(nrow(bos_b_twts_raw)*.02, 0),
           min_docfreq = 2) %>%
  dfm_tfidf()
```

Use quanteda to make sure training and raw data have exact same format as

the training set

```r
# Use quanteda to make sure training and raw data have exact same format
bos_b_twts_apply_dfm <- dfm_select(bos_b_twts_apply_dfm,
                                  pattern = bos_b_twts_train_dfm,
                                  selection = "keep")
bos_b_twts_apply_matrix <- as.matrix(bos_b_twts_apply_dfm)
```

Prepare the data further

```r
# taking the transpose of the singular matrix
sigma.inverse <- 1 / train_lsa$d

# transpose of the term matrix
u.transpose <- t(train_lsa$u)
apply_svd <- t(sigma.inverse * u.transpose %*% t(bos_b_twts_apply_dfm))
apply_svd<-as.matrix(apply_svd)

# Lastly, we can build the apply data frame to feed into our trained machine
# learning model for predictions.
apply_svd <- data.frame(Label = bos_b_twts_raw$label,
                        ReviewLength= nchar(bos_b_twts_raw$text),
                        apply_svd)
```

Make Predictions, apply to the original data as a new column called "preds"

```r
# Now we can make predictions on the apply data set using our trained SVM.
preds <- predict(SVM_RB, apply_svd)
bos_b_twts_raw$preds <- preds
```

Write to a csv with a time stamp, and output the total time the code took to execute

```r
# Get the current time
systime <- format(Sys.time(),"%Y-%m-%d-%H-%M-%S")

# write the original tweets plus the prediction to a csv with a time stamp
write.csv(data.frame(bos_b_twts_raw), file = paste("Boston_B_Tweet_Predictions_", systime, ".csv", sep

# Time it took to run the code from beginning to end
# Note that the packages where pre-loaded
total.time <- Sys.time() - start.time
total.time
```

```
## Time difference of 22.501 secs
```