Practical Introduction to Text Classification

MY560

Blake Miller

14 March 2023

Content warning: This problem makes use of data from a project to automate moderation of toxic speech online. Many comments in this dataset contain hate speech and upsetting content. Please take care as you work on this assignment.

This exercise makes use of replication data for the paper Ex Machina: Personal Attacks Seen at Scale by Ellery Wulczyn, Nithum Thain, and Lucas Dixon. The paper introduces a method for crowd-sourcing labels for personal attacks and then draws several inferences about how personal attacks manifest on Wikipedia Talk Pages. They find that, "the majority of personal attacks on Wikipedia are not the result of a few malicious users, nor primarily the consequence of allowing anonymous contributions from unregistered users." We will use their data and SVM models to identify personal attacks.

Let's start by loading some required packages

```
library(doMC)
library(glmnet)
library(quanteda)
```

Representing Text Features

Preprocessing text with quanteda

Before we can do any type of automated text analysis, we will need to go through several "pre-processing" steps before it can be passed to a statistical model. We'll use the `quanteda` package quanteda here.

The basic unit of work for the `quanteda` package is called a `corpus`, which represents a collection of text documents with some associated metadata. Documents are the subunits of a corpus. You can use `summary` to get some information about your corpus.

```
library(quanteda)
library(quanteda.textplots)
```

```
## Warning in register(): Can't find generic `scale_type` in package ggplot2 to
## register S3 method.
```

```
if (!file.exists('attacks.csv')) {
  download.file('https://github.com/lse-my474/pset_data/raw/main/attacks.csv', 'attacks.csv')
}

texts <- read.csv('attacks.csv', stringsAsFactors=F)
texts$attack <- factor(texts$attack)

corpus <- corpus(texts, text_field="text") # create a corpus
corpus
```

```
## Corpus consisting of 15,000 documents and 2 docvars.
## text1 :
## "which may contain more details"
##
## text2 :
## "Regardless, the point is that I am willing to see what infor..."
##
## text3 :
## "Lede   I'm reverting (again) the additions to the lede on ``..."
##
## text4 :
## "I just came to this page and was wondering why there is no `..."
##
## text5 :
## "It's worth having an illustration.  The Type 2 picture, howe..."
##
## text6 :
## "Naming convention violation part #3    (Tiderolls) The guide..."
##
## [ reached max_ndoc ... 14,994 more documents ]
```

We can then create a tokens object from the corpus using the `tokens` function. This gives us our terms which we will process to create features for our document feature matrix. `tokens` has many useful options (check out `?tokens` for more information).

```
?tokens
toks <- tokens(corpus, remove_punct = TRUE, remove_url=TRUE, verbose=TRUE)
```

```
## Creating a tokens object from a corpus input...
##  ...starting tokenization
##  ...text1 to text10000
##  ...preserving hyphens
##  ...preserving social media tags (#, @)
##  ...segmenting into words
##  ...text10001 to text15000
##  ...preserving hyphens
##  ...preserving social media tags (#, @)
##  ...segmenting into words
##  ...63,321 unique types
##  ...removing separators, punctuation, URLs
##  ...complete, elapsed time: 2.91 seconds.
## Finished constructing tokens from 15,000 documents.
```

```
toks
```

```
## Tokens consisting of 15,000 documents and 2 docvars.
## text1 :
## [1] "which"   "may"     "contain" "more"    "details"
##
## text2 :
```

```
##  [1] "Regardless"  "the"         "point"       "is"          "that"
##  [6] "I"           "am"          "willing"     "to"          "see"
## [11] "what"        "information"
## [ ... and 43 more ]
##
## text3 :
##  [1] "Lede"      "I'm"       "reverting" "again"     "the"       "additions"
##  [7] "to"        "the"       "lede"      "on"        "`"         "`"
## [ ... and 55 more ]
##
## text4 :
##  [1] "I"         "just"      "came"      "to"        "this"      "page"
##  [7] "and"       "was"       "wondering" "why"       "there"     "is"
## [ ... and 37 more ]
##
## text5 :
##  [1] "It's"          "worth"         "having"        "an"            "illustration"
##  [6] "The"           "Type"          "2"             "picture"       "however"
## [11] "is"            "frankly"
## [ ... and 9 more ]
##
## text6 :
##  [1] "Naming"            "convention"        "violation"         "part"
##  [5] "#3"                "Tiderolls"         "The"               "guideline"
##  [9] "Wikipedia:Naming" "conventions"       "use"               "English"
## [ ... and 656 more ]
##
## [ reached max_ndoc ... 14,994 more documents ]
```

Next we can create a document-feature matrix by passing our tokens into the `dfm` function.

```
dfm <- dfm(toks, verbose=TRUE)
```

```
## Creating a dfm from a tokens input...
```

```
##  ...lowercasing
```

```
##  ...found 15,000 documents, 52,069 features
```

```
##  ...complete, elapsed time: 0.299 seconds.
```

```
## Finished constructing a 15,000 x 52,069 sparse dfm.
```

```
dfm
```

```
## Document-feature matrix of: 15,000 documents, 52,069 features (99.91% sparse) and 2 docvars.
##        features
## docs    which may contain more details regardless the point is that
##   text1     1   1       1    1       1          0   0     0  0    0
##   text2     0   0       0    0       0          1   3     1  1    1
##   text3     0   0       0    0       0          0   5     0  1    1
##   text4     0   0       0    0       0          0   1     0  2    0
##   text5     0   0       0    0       0          0   1     0  1    0
##   text6     4   2       0    0       0          0  35     0 10   13
## [ reached max_ndoc ... 14,994 more documents, reached max_nfeat ... 52,059 more features ]
```

The `dfm` will show the count of times each word appears in each document (comment):

```r
dfm[1:5, 1:10]
```

```
## Document-feature matrix of: 5 documents, 10 features (66.00% sparse) and 2 docvars.
##        features
## docs    which may contain more details regardless the point is that
##   text1     1   1       1    1       1          0   0     0  0    0
##   text2     0   0       0    0       0          1   3     1  1    1
##   text3     0   0       0    0       0          0   5     0  1    1
##   text4     0   0       0    0       0          0   1     0  2    0
##   text5     0   0       0    0       0          0   1     0  1    0
```

To stem our documents Stemming relies on the `SnowballC` package's implementation of the Porter stemmer:

```r
toks_stem <- tokens_wordstem(toks)
dfm_stem <- dfm(toks_stem, tolower=TRUE)
dfm_stem
```

```
## Document-feature matrix of: 15,000 documents, 40,477 features (99.89% sparse) and 2 docvars.
##        features
## docs    which may contain more detail regardless the point is that
##   text1     1   1       1    1      1          0   0     0  0    0
##   text2     0   0       0    0      0          1   3     1  1    1
##   text3     0   0       0    0      0          0   5     0  1    1
##   text4     0   0       0    0      0          0   1     0  2    0
##   text5     0   0       0    0      0          0   1     0  1    0
##   text6     4   2       0    0      0          0  35     0 10   13
## [ reached max_ndoc ... 14,994 more documents, reached max_nfeat ... 40,467 more features ]
```

```r
example <- tolower(texts$text[5])
tokens(example)
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "it's"          "worth"         "having"        "an"            "illustration"
##  [6] "."             "the"           "type"          "2"             "picture"
## [11] ","             "however"
## [ ... and 16 more ]
```

```r
tokens_wordstem(tokens(example))
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "it"      "worth"   "have"    "an"      "illustr" "."       "the"
##  [8] "type"    "2"       "pictur"  ","       "howev"
## [ ... and 16 more ]
```

In a large corpus like this, many features often only appear in one or two documents. In some case it's a good idea to remove those features, to speed up the analysis or because they're not relevant. We can `trim` the dfm:

```r
dfm_trimmed <- dfm_trim(dfm_stem, min_docfreq=75, verbose=TRUE)
```

```
## Removing features occurring:

##   - in fewer than 75 documents: 39,363

##   Total features removed: 39,363 (97.2%).
```

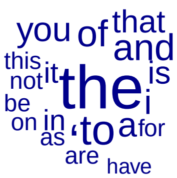
```r
dfm_trimmed
```

```
## Document-feature matrix of: 15,000 documents, 1,114 features (96.95% sparse) and 2 docvars.
```

```
##        features
## docs    which may contain more detail regardless the point is that
##   text1     1   1       1    1      1          0   0     0  0    0
##   text2     0   0       0    0      0          1   3     1  1    1
##   text3     0   0       0    0      0          0   5     0  1    1
##   text4     0   0       0    0      0          0   1     0  2    0
##   text5     0   0       0    0      0          0   1     0  1    0
##   text6     4   2       0    0      0          0  35     0 10   13
## [ reached max_ndoc ... 14,994 more documents, reached max_nfeat ... 1,104 more features ]
```

It's often a good idea to take a look at a wordcloud of the most frequent features to see if there's anything weird.

```
textplot_wordcloud(dfm_trimmed, rotation=0, min_size=.75, max_size=3, max_words=20)
```



What is going on? We probably want to remove words and symbols which are not of interest to our data, such as http here. This class of words which is not relevant are called stopwords. These are words which are common connectors in a given language (e.g. "a", "the", "is"). We can also see the list using `topFeatures`

```
topfeatures(dfm_trimmed, 25)
```

```
##    the      `     to    and     of      a    you      i     is   that     it
##  46609  33161  27853  22411  21566  20691  19974  19142  17196  15527  15237
##     in    for    not   this     be     on    are     as   have articl   with
##  13964   9314   9253   9199   9137   8342   7475   7378   7053   6357   5853
##   your    was     if
##   5394   5348   5162
```

We can remove twitter words and stopwords using `tokens_remove()`:

```
toks_stop <- tokens_remove(toks_stem, stopwords("english"))
?tokens_remove

dfm_stop <- dfm(toks_stop)
textplot_wordcloud(dfm_stop, rotation=0, min_size=.5, max_size=5, max_words=20)
```

Basic Text Classification

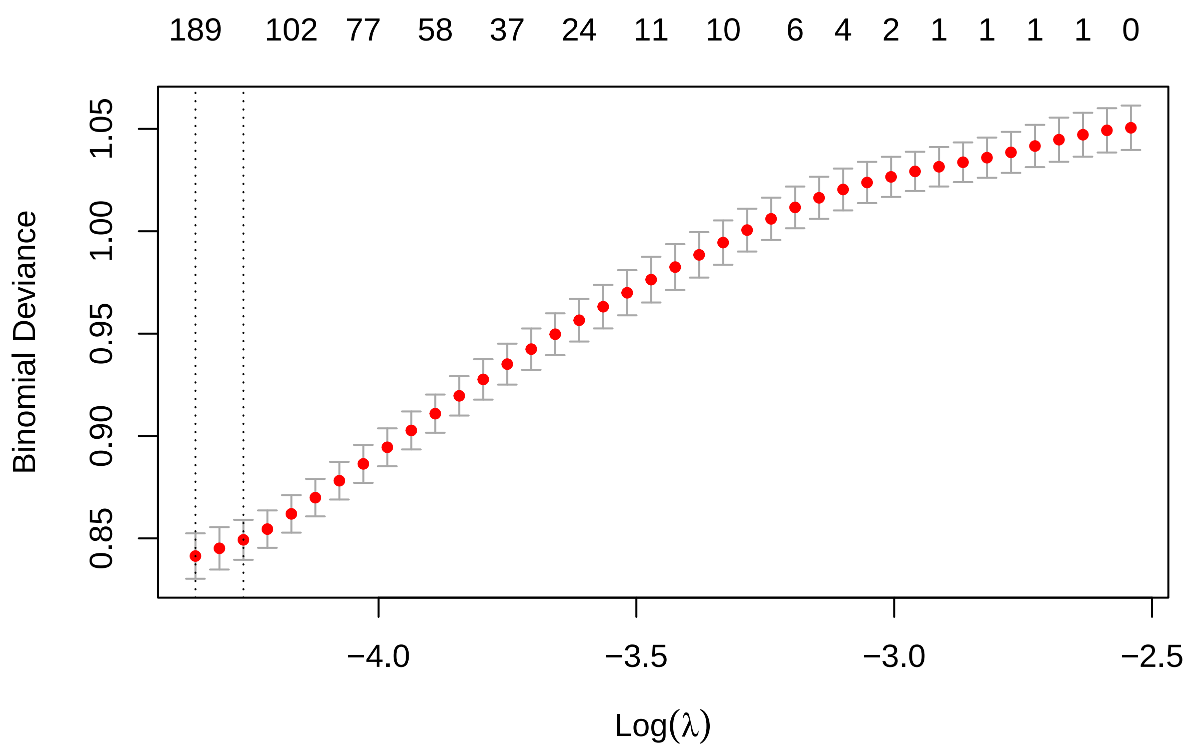
```
# Separate labeled documents from unlabeled documents
unlabeled <- dfm_subset(dfm, is.na(texts$attack))
labeled <- dfm_subset(dfm, !is.na(texts$attack))

N <- nrow(labeled)

tr <- sample(1:N, floor(N*.8)) # indexes for test data
```

Let's train a logistic regression (family="binomial") with a LASSO penalty. We choose the optimal value of lambda using cross-validation with `cv.glmnet`. Using `plot`, we can plot error (binomial deviance) for all values of λ chosen by `cv.glmnet`. How many non-zero coefficients are in the model where misclassification error is minimized? How many non-zero coefficients are in the model one standard deviation from where misclassification error is minimized?

```
registerDoMC(cores=5) # trains all 5 folds in parallel (at once rather than one by one)
mod <- cv.glmnet(labeled[tr,], docvars(labeled,"attack")[tr], nfolds=5, parallel=TRUE, family="binomial
```

```
## Warning: from glmnet Fortran code (error code -41); Convergence for 41th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
plot(mod)
```



According to cross-validation error calculated by `cv.glm`, we can examine the optimal λ stored in the output? We can then find the corresponding CV error for this value of λ.

```
mod$lambda.min
```

```
## [1] 0.01284145
```

```
log(mod$lambda.min) # To match the axis in the plot above
```

```
## [1] -4.355077
```

```
lam_min <- which(mod$lambda == mod$lambda.min)
lam_min
```

```
## [1] 40
```

```
cv_min <- mod$cvm[lam_min]
cv_min
```

```
## [1] 0.841382
```

Error Measures

We can evaluate test set performance for the best-fit model using accuracy.

```
pred_min <- predict(mod, labeled[-tr,], s="lambda.min", type="class")
mean(pred_min == labeled$attack[-tr])
```

```
## [1] 0.810596
```

```
lam_1se <- which(mod$lambda == mod$lambda.1se)
pred_1se <- predict(mod, labeled[-tr,], s="lambda.1se", type="class")
mean(pred_1se == labeled$attack[-tr])
```

```
## [1] 0.8072848
```

We can also examine the confusion matrix to get a better idea of the error. We can also use this confusion matrix to calculate other error measures using the functions specified below.

```
table(pred_min, labeled$attack[-tr])
```

```
##
## pred_min    0    1
##        0 1149  273
##        1   13   75
```

```
table(pred_1se, labeled$attack[-tr])
```

```
##
## pred_1se    0    1
##        0 1151  280
##        1   11   68
```

```
## function to compute accuracy
accuracy <- function(ypred, y){
    tab <- table(ypred, y)
    return(sum(diag(tab))/sum(tab))
}
# function to compute precision
precision <- function(ypred, y){
    tab <- table(ypred, y)
    return((tab[2,2])/(tab[2,1]+tab[2,2]))
}
# function to compute recall
recall <- function(ypred, y){
    tab <- table(ypred, y)
    return(tab[2,2]/(tab[1,2]+tab[2,2]))
}

accuracy(pred_min, labeled$attack[-tr])
```

```
## [1] 0.810596
```

```
precision(pred_min, labeled$attack[-tr])
```

```
## [1] 0.8522727
```

```
recall(pred_min, labeled$attack[-tr])
```

```
## [1] 0.2155172
```

```
accuracy(pred_1se, labeled$attack[-tr])
```

```
## [1] 0.8072848
```

```
precision(pred_1se, labeled$attack[-tr])
```

```
## [1] 0.8607595
```

```
recall(pred_1se, labeled$attack[-tr])
```

```
## [1] 0.1954023
```

Using the model we have identified with the minimum CV error, we can also look at the largest and smallest coefficient estimates and the features associated with them.

```
beta <- mod$glmnet.fit$beta[,lam_min]
ind <- order(beta)

head(beta[ind], n=10)
```

```
##       chad      thank    welcome     cheers    thought    article   redirect
## -0.5306120 -0.3995257 -0.2111662 -0.1896710 -0.1672529 -0.1614550 -0.1589753
##       best        may     please
## -0.1267264 -0.1212465 -0.1089461
```

```
tail(beta[ind], n=10)
```

```
##   fuckin  faggots    sucks   fucked     cock    idiot     scum     piss
## 1.293629 1.311311 1.341774 1.368173 1.402995 1.423979 1.425971 1.490467
## assholes  fucking
## 1.529343 1.563344
```

Active Learning

In the labeled data, we can see that there is a slight class imbalance.

```
table(labeled$attack)
```

```
##
##    0    1
## 5873 1677
```

For this example, we will select the next batch of documents to label using **uncertainty sampling**. Uncertainty sampling involves selecting an observation for labeling based on a measure of the uncertainty of a model's class prediction for that observation. This measure of uncertainty can come in many forms, but for the sake of familiarity, we will use **logistic regression**.

The predicted probabilities from a logistic regression model can be used as a measure of *model uncertainty* about the label of each observation in our *unlabeled* data. The logistic regression classifier will be most uncertain when the predicted probability is .5. In this scenario, the classifier is indifferent as to whether the observation is positive or negative. To sample 20 unlabeled observations using this form of active learning, we would *query* or *select* observations for a human to label where $\hat{p}$ is closest to .5 (i.e. $|\hat{p} - .5|$).

```
nrow(labeled)
```

```
## [1] 7550
```

```
nrow(unlabeled)
```

```
## [1] 7450
```

```
pred <- predict(mod, unlabeled, type="response") # predicted probabilities
sorted <- sort(abs(pred - .5), decreasing=FALSE, index.return=TRUE)
pred[head(sorted$ix)] # Predicted probabilities closest to .5
```

```
## [1] 0.5001330 0.4995642 0.5004368 0.5008669 0.4985843 0.5022433
```

```
head(sorted$x) # Distance from .5
```

```
## [1] 0.0001329867 0.0004358386 0.0004368090 0.0008668979 0.0014156788
## [6] 0.0022432812
```

```
to_label <- docvars(unlabeled[sorted$ix[1:10],],"id")
to_label
```

```
##  [1] 576742314 277461942 364159183  38517702 284894163 574221731 420043881
##  [8]  69237677 607823536  76347088
```

```
texts[texts$id %in% to_label, "text"] # Our sample to label
```

```
##  [1] "Who are you calling a vandal?? You belong to a gang of Catholics (being a Mexican why the heck
##  [2] "Are you a pole smoker? Do you suck cock?  -Preceding unsigned comment added by"
##  [3] "Dont revert my edits to my fuckin userpage. If you want her organs you go some else where unna
##  [4] "Wow  \n\nHoly fuck this girl is hot as hell. I know we aren't supposed to put things like this
##  [5] "hey ricboom who ever the fuck you are if you go to my schoolim gonna beat the shit out of you"
##  [6] "Evidently it is one rule for unfavorable accounts and another for bitches like Katieh5758 who 
##  [7] "Once again, I disagree on all counts. It should be PATENTLY obviously to anyone reading about 
##  [8] "How big are Amanda Tapping's boobs   \nHow big are Amanda Tapping's boobs?."
##  [9] ", 19 April 2009 (UTC)\n:::::::::Stay out of it damn it, do not fuck around with me. Just go so
## [10] "please refrain from your islamofascist garbage. the article on BNF has nothing to do with BJP.
```

Once we add labels to these documents, we would refit the model with them in the `labeled` set, and repeat the process above to query another batch of documents.

Sentiment analysis using LASSO

Sentiment analysis is a method for measuring the positive or negative valence of language. In this problem, we will use movie review data to create scale of negative to positive sentiment ranging from 0 to 1.

In this exercise, we will do this using a logistic regression model with ℓ_1 penalty (the lasso) trained on a corpus of 25,000 movie reviews from IMDB.

First, lets install and load packages.

```
#install.packages("doMC", repos="http://R-Forge.R-project.org")
#install.packages("glmnet")
#install.packages("quanteda")
#install.packages("readtext")

library(doMC)
library(glmnet)
library(quanteda)
library(readtext)
```

In this first block, I have provided code that downloads the preprocessed data into a matrix of term counts (columns) for each document (rows). This matrix is named `dfm`. Each document is labeled 0 or 1 in the document variable `sentiment`: positive or negative sentiment respectively.

```
options(timeout=max(300, getOption("timeout")))
download.file("https://github.com/lse-my474/pset_data/raw/main/12500_dtm.rds", "12500_dtm.rds")
```

```r
download.file("https://github.com/lse-my474/pset_data/raw/main/6250_dtm.rds", "6250_dtm.rds")
download.file("https://github.com/lse-my474/pset_data/raw/main/3125_dtm.rds", "3125_dtm.rds")
```

Below is starter code to help you properly train a lasso model using the `.rds` files generated in the previous step. As you work on this problem, it may be helpful when troubleshooting or debugging to reduce `nfolds` to 3 or change N to either 3125 or 6250 to reduce the time it takes you to run code. You can also choose a smaller N if your machine does not have adequate memory to train with the whole corpus.

```r
# change N to 3125 or 6250 if computation is taking too long
N <- 12500

dfm <- readRDS(paste(N, "_dtm.rds", sep=""))
dfm$id <- 1:nrow(dfm)
tr <- sample(dfm$id, floor(nrow(dfm)*.8)) # indexes for training data

registerDoMC(cores=5) # trains all 5 folds in parallel (at once rather than one by one)

mod <- cv.glmnet(dfm_subset(dfm, id %in% tr), dfm_subset(dfm, id %in% tr)$sentiment,
                 nfolds=5, parallel=TRUE, family="binomial")
```

a. Plot misclassification error for all values of λ chosen by `cv.glmnet`. How many non-zero coefficients are in the model where misclassification error is minimized? How many non-zero coefficients are in the model one standard deviation from where misclassification error is minimized? Which model is sparser?

b. According to the estimate of the test error obtained by cross-validation, what is the optimal λ stored in your `cv.glmnet()` output? What is the CV error for this value of λ? *Hint: The vector of λ values will need to be subsetted by the index of the minimum CV error.*

c. What is the test error for the λ that minimizes CV error? What is the test error for the 1 S.E. λ? How well did CV error estimate test error?

d. Using the model you have identified with the minimum CV error, identify the 10 largest and the 10 smallest coefficient estimates and the features associated with them. Do they make sense? Do any terms look out of place or strange? In 3-5 sentences, explain your observations. *Hint: Use `order()`, `head()`, and `tail()`. The argument `n=10` in the `head()`, and `tail()` functions will return the first and last 10 elements respectively.*