Text classification in R

Jack Lovell

11/2/2020

Getting closer with meta-analytic database

In the spirit of the final project, I am going to shift my attention to the problem of text classification for this report. The problem I stated in week 1 has still yet to be addressed, can we save time and avoid errors by using text classification in the paper selection stage of a meta-analysis. This is an outstanding problem that is recieving increasing attention in the field. As discussed earlier, an ideal algorithm will not remove the article from the database but rather flag it if it is indeed what we are interested in. We are getting closer by the day to having a fully annotated datbase, but still have some work to do. Due to us coming increasingly closer to a finalized dataset, I have decided to learn about text classification in R. I will closely follow the tutorial detailed in this walkthrough: https://cfss.uchicago.edu/notes/supervised-text-classification/ Then use an ope-source dataset to practice the skills I have learned!

Text classification in R

Typically machine learning, and in particular deep learning, are done in the programming language Python. Due to the recent explosion in popularity of these subjects, certain packages in R allow you to do similar modeling. One application of machine learning is for a machine to learn what aspects of text are associated with a specific label. This is often known as *supervised learning*. In a supervised learning algorithm, we often have a training dataset and a testing dataset. Within the training dataset you typically have certain data you are interested in classifying into groups. Those data are labeled by their respective groups, and the algorithm learns what makes data within a group similar. You then can test how well your model performs on a set of data that the model has never seen. It will in turn label the data, and you can see how much those labels match the ground truth! For a schematic of this see the graphic below.

[]!(https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/07/Supervised-Learning-in-ML-tv.jpg)

US Congress data

Using the power of R and the tidyverse libraries, we can use an open dataset of hand-labeled US Congress bills to begin our education.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidytext)
library(stringr)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(remotes)
remotes::install_github("uc-cfss/rcfss")
```

```
## Skipping install of 'rcfss' from a github remote, the SHA1 (24dca66a) has not changed since last ins
##   Use `force = TRUE` to force installation
```

```
set.seed(1234)
theme_set(theme_minimal())

# get USCongress data
data(USCongress, package = "rcfss")

# topic labels
major_topics <- tibble(
  major = c(1:10, 12:21, 99),
  label = c("Macroeconomics", "Civil rights, minority issues, civil liberties",
            "Health", "Agriculture", "Labor and employment", "Education", "Environment",
            "Energy", "Immigration", "Transportation", "Law, crime, family issues",
            "Social welfare", "Community development and housing issues",
            "Banking, finance, and domestic commerce", "Defense",
            "Space, technology, and communications", "Foreign trade",
            "International affairs and foreign aid", "Government operations",
```

```
            "Public lands and water management", "Other, miscellaneous")
)

(congress <- as_tibble(USCongress) %>%
    mutate(text = as.character(text)) %>%
    left_join(major_topics))
```

```
## Joining, by = "major"
```

```
## # A tibble: 4,449 x 7
##       ID  cong billnum h_or_sen major text                         label
##    <dbl> <dbl>   <dbl> <chr>    <dbl> <chr>                        <chr>
##  1     1   107    4499 HR          18 To suspend temporarily the~ Foreign trade
##  2     2   107    4500 HR          18 To suspend temporarily the~ Foreign trade
##  3     3   107    4501 HR          18 To suspend temporarily the~ Foreign trade
##  4     4   107    4502 HR          18 To reduce temporarily the ~ Foreign trade
##  5     5   107    4503 HR           5 To amend the Immigration a~ Labor and emp~
##  6     6   107    4504 HR          21 To amend title 38, United ~ Public lands ~
##  7     7   107    4505 HR          15 To repeal subtitle B of ti~ Banking, fina~
##  8     8   107    4506 HR          18 To suspend temporarily the~ Foreign trade
##  9     9   107    4507 HR          18 To suspend temporarily the~ Foreign trade
## 10    10   107    4508 HR          18 To suspend temporarily the~ Foreign trade
## # ... with 4,439 more rows
```

Ok so it looks like we have a large dataset of bills, that are either in the house of representatives or senate. Each bill has a topic associated with it which is really nice. One problem we may be concerned with is classifying each bill into it's respective category! Using R we can begin to do this, but before we do any sort of modeling we need to create a tidy text data frame.

Tidy text dataframe

A tidy text dataframe will allow us to analyze and view our data more efficiently. Let's use some of the preprocessing techniques we learned from the earlier weeks of the semester.

```
(congress_tokens <- congress %>%
   unnest_tokens(output = word, input = text) %>%
   # remove numbers
   filter(!str_detect(word, "^[0-9]*$")) %>%
   # remove stop words
   anti_join(stop_words) %>%
   # stem the words
   mutate(word = SnowballC::wordStem(word)))
```

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

```
## # A tibble: 58,820 x 7
##       ID  cong billnum h_or_sen major label         word
##    <dbl> <dbl>   <dbl> <chr>    <dbl> <chr>         <chr>
##  1     1   107    4499 HR          18 Foreign trade suspend
##  2     1   107    4499 HR          18 Foreign trade temporarili
##  3     1   107    4499 HR          18 Foreign trade duti
```

```
##  4     1   107    4499 HR          18 Foreign trade fast
##  5     1   107    4499 HR          18 Foreign trade magenta
##  6     1   107    4499 HR          18 Foreign trade stage
##  7     2   107    4500 HR          18 Foreign trade suspend
##  8     2   107    4500 HR          18 Foreign trade temporarili
##  9     2   107    4500 HR          18 Foreign trade duti
## 10     2   107    4500 HR          18 Foreign trade fast
## # ... with 58,810 more rows
```

Ok great, now that we've unnested and tokenized Congress we can begin to model our data! These reason we did this was is to decrease the number of features our model has to learn from. The more features, the more complex the model. Now, we still need to transform our data a bit, as this has a tok-per-row format, where we need a row-per-document matrix.

```
(congress_dtm <- congress_tokens %>%
   # get count of each token in each document
   count(ID, word) %>%
   # create a document-term matrix with all features and tf weighting
   cast_dtm(document = ID, term = word, value = n))
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 4902)>>
## Non-/sparse entries: 55033/21753965
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency (tf)
```

Weighting

Ok great! We now have document-per-row dataframe. The weighting that is typically done is based of how often a term occurs within a text. Let's use **tfidf** which is the frequency of a term adjusted for how rarely it is used.

```
congress_tokens %>%
  # get count of each token in each document
  count(ID, word) %>%
  # create a document-term matrix with all features and tf-idf weighting
  cast_dtm(document = ID, term = word, value = n,
           weighting = tm::weightTfIdf)
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 4902)>>
## Non-/sparse entries: 55033/21753965
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

We could compare how each model performs using each weighting, but for now we will just continue with tfidf

Sparsity

We can also remove sparse terms from our set of features to improve our model. Removing sparse features in this sense means removing tokens that do not appear across many documents.

```
(congress_dtm <- removeSparseTerms(congress_dtm, sparse = .99))
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 209)>>
## Non-/sparse entries: 33794/896047
## Sparsity           : 96%
## Maximal term length: 11
## Weighting          : term frequency (tf)
```

EDA

Now let's check out our data! First we will want to calculate the tf-idf of each term, in which we treat each major topic code as the document. This is differnet than before as it is not tf-idf by each individual bill.

```
(congress_tfidf <- congress_tokens %>%
   count(label, word) %>%
   bind_tf_idf(term = word, document = label, n = n))
```

```
## # A tibble: 13,190 x 6
##    label       word        n       tf   idf    tf_idf
##    <chr>       <chr>   <int>    <dbl> <dbl>     <dbl>
##  1 Agriculture abund       2 0.00106  3.00  0.00317
##  2 Agriculture access      1 0.000529 0.163 0.0000860
##  3 Agriculture account     1 0.000529 0.223 0.000118
##  4 Agriculture acet        2 0.00106  2.30  0.00244
##  5 Agriculture acid        2 0.00106  1.90  0.00201
##  6 Agriculture acreag      1 0.000529 2.30  0.00122
##  7 Agriculture act        59 0.0312   0     0
##  8 Agriculture action      5 0.00265  0.598 0.00158
##  9 Agriculture activ       2 0.00106  0.223 0.000236
## 10 Agriculture actual      1 0.000529 1.90  0.00100
## # ... with 13,180 more rows
```

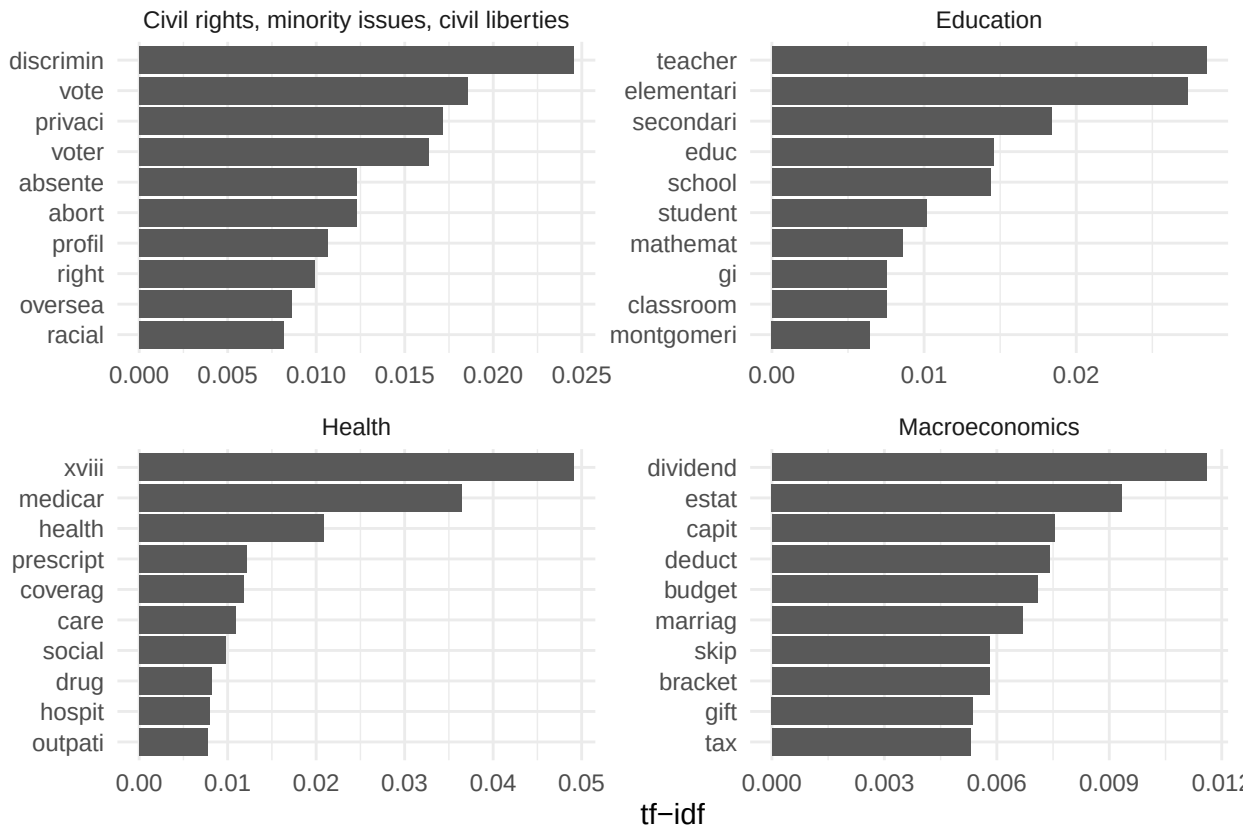
Now lets plot this! It will be for 20 categories, so let's just look at 4 of them

```
# sort the data frame and convert word to a factor column
plot_congress <- congress_tfidf %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

# graph the top 10 tokens for 4 categories
plot_congress %>%
  filter(label %in% c("Macroeconomics",
                      "Civil rights, minority issues, civil liberties",
                      "Health", "Education")) %>%
  group_by(label) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder_within(word, tf_idf, label)) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col() +
  scale_x_reordered() +
```

```
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~ label, scales = "free") +
  coord_flip()
```

```
## Selecting by tf_idf
```



Modeling

Now let's build our model! We will use a random forest tree first. Note that we convert x to a traditional matrix, and we return to the original congress dataframe. This is because our x would be a special type of matrix if we did not transform it back to an original matrix. Also the method ranger is used because it is faster.

```
congress_rf <- train(x = as.matrix(congress_dtm),
                     y = factor(congress$major),
                     method = "ranger",
                     num.trees = 200,
                     trControl = trainControl(method = "oob"))
```

```
## Growing trees.. Progress: 51%. Estimated remaining time: 30 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 23 seconds.
## Growing trees.. Progress: 77%. Estimated remaining time: 9 seconds.
## Growing trees.. Progress: 10%. Estimated remaining time: 4 minutes, 55 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 2 minutes, 47 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 35 seconds.
```

We can also train with a different number of trees

```
# some documents are lost due to not having any relevant tokens after tokenization
# make sure to remove their associated labels so we have the same number of observations
congress_slice <- slice(congress, as.numeric(congress_dtm$dimnames$Docs))

library(tictoc)

tic()
congress_rf_10 <- train(x = as.matrix(congress_dtm),
                        y = factor(congress_slice$major),
                        method = "ranger",
                        num.trees = 10,
                        importance = "impurity",
                        trControl = trainControl(method = "oob"))
toc()
```

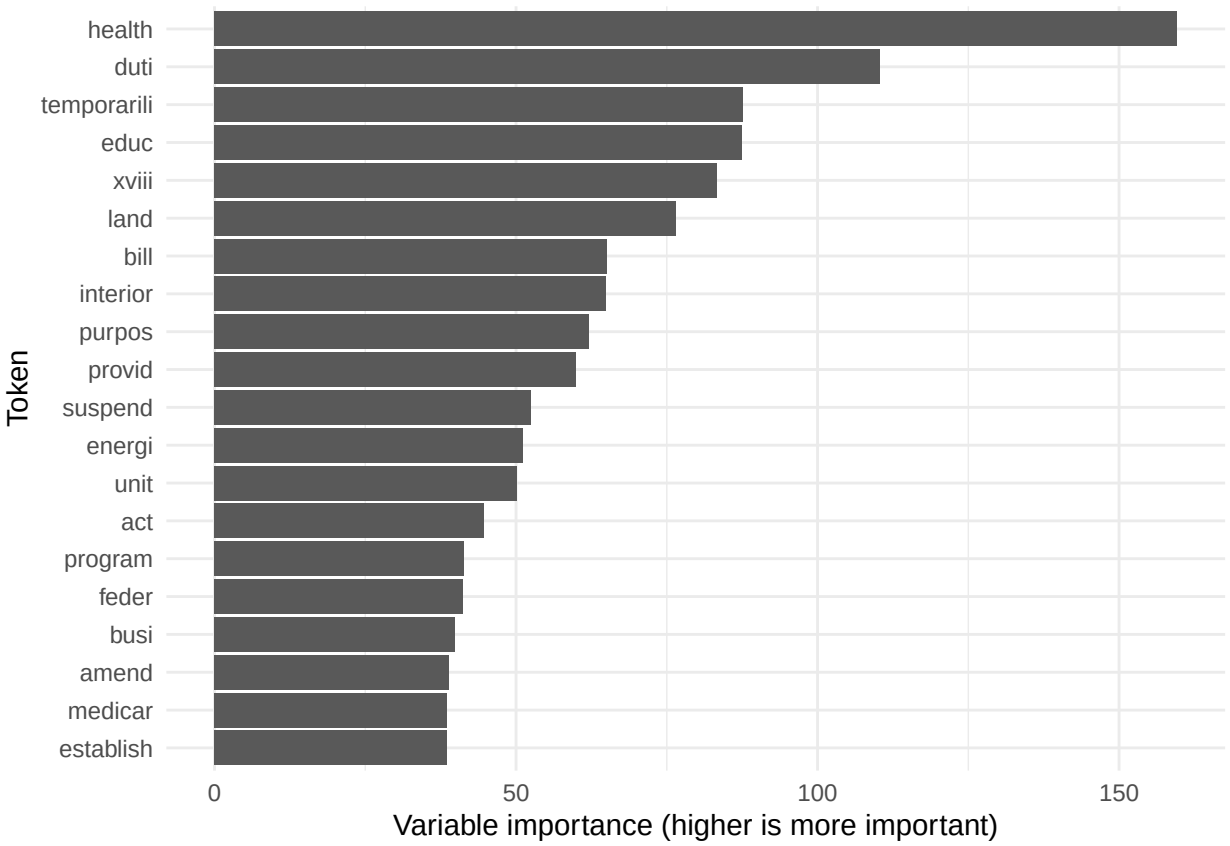
```
## 7.45 sec elapsed
```

```
tic()
congress_rf_200 <- train(x = as.matrix(congress_dtm),
                         y = factor(congress_slice$major),
                         method = "ranger",
                         num.trees = 200,
                         importance = "impurity",
                         trControl = trainControl(method = "oob"))
toc()
```

```
## 94.792 sec elapsed
```

That took way longer with 200 trees! But let's take a look at the model performance. It is more typical to have around 200 trees so let's look into this.

```
congress_rf_200$finalModel %>%
  # extract variable importance metrics
  ranger::importance() %>%
  # convert to a data frame
  enframe(name = "variable", value = "varimp") %>%
  top_n(n = 20, wt = varimp) %>%
  # plot the metrics
  ggplot(aes(x = fct_reorder(variable, varimp), y = varimp)) +
  geom_col() +
  coord_flip() +
  labs(x = "Token",
       y = "Variable importance (higher is more important)")
```



Great!It looks like the most important variable is health (not wrong!). If we had some testing data we could test how well this model performs on data without any sort of label, but that will be saved for next week, as testing your algorithm is it's own rabbit hole we can go down!