607 Project 4

Lu Beyer

INSTRUCTIONS

It can be useful to be able to classify new "test" documents using already classified "training" documents. A common example is using a corpus of labeled spam and ham (non-spam) e-mails to predict whether or not a new document is spam.

For this project, you can start with a spam/ham dataset, then predict the class of new documents (either withheld from the training dataset or from another source such as your own spam folder).

I downloaded and used a sms spam dataset collected by the UC Irvine Machine Learning Repository: https://archive.ics.uci.edu/dataset/228/sms+spam+collection

First we load our libraries, and then load our data through read.table(), setting sep "" to note that records are separated by tabs. We then rename and select our relevant columns for use.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tm)
```

```
## Warning: package 'tm' was built under R version 4.3.3
```

```
## Loading required package: NLP
##
## Attaching package: 'NLP'
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(SnowballC)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```r
#load data
data <- read.table("SMSSpamCollection", sep = "\t", stringsAsFactors = FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,
## : EOF within quoted string
```

```r
#renaming and selecting relevant columns
a <- data %>%
  rename(txt = V2,
         flag = V1) %>%
  select(txt, flag)
```

Using tm::Corpus, we convert our data into a corpus.

```r
#converting data to corpus
text_corpus <- Corpus(VectorSource(a$txt), readerControl = list(language = "en", encoding = "UTF-8"))
text_corpus
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1779
```

tm_map allows us to perform various standard text cleaning operations, such as removing stopwords, whitespaces, and stemming, which is reduces a word to its root. ie: "running" would be reduced to "run", for easier, more consistent text parsing.

```r
#removing whitespace, converting text to lowercase, removing stopwords,
#removing punctionation, numbers, and stemming words within corpus
b <- text_corpus %>%
  tm_map(stripWhitespace) %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removeWords, stopwords("english")) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeNumbers) %>%
  tm_map(stemDocument)
```

```
## Warning in tm_map.SimpleCorpus(., stripWhitespace): transformation drops
## documents
```

```
## Warning in tm_map.SimpleCorpus(., content_transformer(tolower)): transformation
## drops documents
```

```
## Warning in tm_map.SimpleCorpus(., removeWords, stopwords("english")):
## transformation drops documents
```

```
## Warning in tm_map.SimpleCorpus(., removePunctuation): transformation drops
## documents
```

```
## Warning in tm_map.SimpleCorpus(., removeNumbers): transformation drops
## documents

## Warning in tm_map.SimpleCorpus(., stemDocument): transformation drops documents
```

Using tm to create a Document Term Matrix (dtm)

```
#create document term matrix using tm
dtm <- DocumentTermMatrix(b)

dtm
```

```
## <<DocumentTermMatrix (documents: 1779, terms: 6610)>>
## Non-/sparse entries: 33663/11725527
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

We then set our sample size to 75% of the records to train our model on, and identify those records.

```
#set sample size, set indices
samp <-  round(0.75 * nrow(dtm))
indices_train <- sample(seq_len(nrow(dtm)), size = samp)
```

```
#split data based on indices
train_data <-  dtm[indices_train, ]
test_data  <-  dtm[-indices_train, ]
```

```
#extract labels based on indices
train_label <-  a[indices_train, ]$flag
test_label  <-  a[-indices_train, ]$flag
```

We then use the identified training data to train our model

```
#train model
nb <- naiveBayes(as.matrix(train_data), as.factor(train_label))
```

Using our trained model, we predict the the results of the remaining 25% of records

```
#predict new test data labels based on trained model
result <- predict(nb, newdata = as.matrix(test_data))
```

```
#create and print confusion matrix based on test results
cm <- table(Predicted = result, Actual = test_label)
print(cm)
```

```
##          Actual
## Predicted ham spam
##      ham    0    0
##      spam 395   50
```