Emilio Sayun

2024-05-21

read the imdb data from webpage

```
imdb <- read.table("http://www.econ.upf.edu/~michael/MMR/imdb_labelled.tx
t",
                   col.names=c("comment","positive_flag"),
                   quote="", comment.char="", sep="\t",
                   stringsAsFactors=FALSE)
```

some extra options in above read.table function:

quote=“” means there are no quotes around the character strings

stringsAsFactors=FALSE does not automatically convert strings to factors

```
dim(imdb)
```

```
## [1] 1000    2
```

add an ID number of the movie reviews

```
imdb$review_id <- 1:nrow(imdb)
```

look at a bit of the data: notice the classification is given, but we

we are going to try to predict the classification

```
#View(imdb)
table(imdb$positive_flag)
```

```
##
##   0   1
## 500 500
```

tokenize using unnest_tokens

```
require(tidytext)
```

```
## Loading required package: tidytext
```

for the pipe function, we need the ‘dplyr’ package

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

tidy_imdb <- imdb %>% unnest_tokens(word, comment)
head(tidy_imdb, 20)

##    positive_flag review_id       word
## 1              0         1          a
## 2              0         1       very
## 3              0         1       very
## 4              0         1       very
## 5              0         1       slow
## 6              0         1     moving
## 7              0         1    aimless
## 8              0         1      movie
## 9              0         1      about
## 10             0         1          a
## 11             0         1 distressed
## 12             0         1   drifting
## 13             0         1      young
## 14             0         1        man
## 15             0         2        not
## 16             0         2       sure
## 17             0         2        who
## 18             0         2        was
## 19             0         2       more
## 20             0         2       lost

dim(tidy_imdb)

## [1] 14472     3
```

###— Comment: tidy_imdb contains a row for every single word in all 1000 reviews

We have to first remove the stopwords and you have to identify the unique words

```
movie_counts <- tidy_imdb %>% count(word, sort=TRUE)
```

number of unique words

```
dim(movie_counts)

## [1] 3125    2
```

```
head(movie_counts)
```

```
##   word   n
## 1  the 848
## 2  and 434
## 3    a 433
## 4   of 377
## 5   is 340
## 6 this 292
```

###— Question 1: What is in the rows of the above partial list ###— and how many rows does it have?

It contains a unique word from the IMDb movie reviews and the frequency count of how many times that word appeared across all reviews. In this case the list is order from the word with highest frequency which is "the"

It has 3125 rows.

removal of stop words

these are the ones that tidytext uses:

```
#View(stop_words)
movie_words <- tidy_imdb %>% anti_join(stop_words, by = "word")
dim(movie_words)
```

```
## [1] 5523    3
```

```
head(movie_words)
```

```
##   positive_flag review_id       word
## 1             0         1       slow
## 2             0         1     moving
## 3             0         1    aimless
## 4             0         1      movie
## 5             0         1 distressed
## 6             0         1   drifting
```

###— Question 2: What is in the rows of the above partial list ###— and how many rows does it have?

Each row in the list contains a single word from the IMDb movie reviews after removing stop words, along with the sentiment label of the review (positive_flag column 0 for negative, 1 for positive) and the unique identifier for the review review_id column.

it has 5523 rows after removing common stop words

```
movie_counts <- movie_words %>% count(word, sort=TRUE)
dim(movie_counts)
```

```
## [1] 2649    2
```

```
head(movie_counts)
```

```
##         word   n
## 1      movie 181
## 2       film 160
## 3        bad  71
## 4     acting  43
## 5       time  43
## 6 characters  35
```

###— Question 3: What is in the rows of the above partial list ###— and how many rows does it have?

Each row in the list contains a unique word from the IMDb movie reviews after removing the stop words and the frequency count of how many times that word appeared across all reviews.

It has 2649 rows

Lemmatizing the tokenized data frame

```
require(textstem)
```

```
## Loading required package: textstem
```

```
## Loading required package: koRpus.lang.en
```

```
## Loading required package: koRpus
```

```
## Loading required package: sylly
```

```
## For information on available language packages for 'koRpus', run
##
##   available.koRpus.lang()
##
## and see ?install.koRpus.lang()
```

```
movie_words <- movie_words %>%
  mutate(lemma = lemmatize_words(word))
dim(movie_words)
```

```
## [1] 5523    4
```

```
head(movie_words)
```

```
##   positive_flag review_id       word    lemma
## 1             0         1       slow     slow
```

```
## 2             0         1     moving     move
## 3             0         1    aimless  aimless
## 4             0         1      movie    movie
## 5             0         1 distressed distress
## 6             0         1   drifting    drift
```

###— Question 4: What is in the rows of the above partial list ###— and how many rows does it have?

Each row in the list contains the sentiment label of the review (`positive_flag`), the unique identifier for the review (`review_id`), the original word from the movie reviews after removing stop words (`word`), and the lemmatized form of the original word (`lemma`). Lemmatization is the process of converting a word to its base or dictionary form.

It has 5523 rows

```
movie_counts <- movie_words %>% count(lemma, sort=TRUE)
dim(movie_counts)

## [1] 2324    2
```

```
head(movie_counts)

##       lemma   n
## 1     movie 210
## 2      film 186
## 3       bad  93
## 4 character  58
## 5      time  49
## 6       act  48
```

###— Question 5: What is in the rows of the above partial list ###— and how many rows does it have? ###————————————————————————-

Each row in the list contains a unique lemmatized word from the IMDb movie reviews and the frequency count of how many times that lemmatized word appeared across all reviews.

It has 2324 rows

```
total_words <- movie_words %>%
  count(review_id, word) %>%
  group_by(review_id) %>% summarize(total = sum(n))
dim(total_words)

## [1] 991   2
```

```r
head(total_words)
```

```
## # A tibble: 6 × 2
##   review_id total
##       <int> <int>
## 1         1     6
## 2         2     6
## 3         3    15
## 4         4     2
## 5         5     6
## 6         6     9
```

###— Question 6: What is in the rows of the above partial list ###— and why are there only 991 rows and not 1000? ###————————————————————————————-

Each row in the partial list contains the unique identifier for each movie review and the total number of words excluding stop words in that review.

It has 991 rows

There are only 991 rows because some movie reviews might have contained only stop words, which were removed.

```r
words_grouped <- movie_words%>%
  count(review_id, word) %>%
  left_join(total_words)
```

```
## Joining with `by = join_by(review_id)`
```

```r
words_tfidf <- words_grouped %>%
  bind_tf_idf(review_id, word, n)
dim(words_tfidf)
```

```
## [1] 5412    7
```

```r
head(words_tfidf)
```

```
##   review_id       word n total          tf      idf     tf_idf
## 1         1    aimless 1     6 1.000000000 6.090178 6.09017802
## 2         1 distressed 1     6 1.000000000 6.090178 6.09017802
## 3         1   drifting 1     6 1.000000000 6.090178 6.09017802
## 4         1      movie 1     6 0.005524862 6.090178 0.03364739
## 5         1     moving 1     6 0.250000000 6.090178 1.52254451
## 6         1       slow 1     6 0.166666667 6.090178 1.01502967
```

[1] 5421 7 # review_id word n total tf idf tf_idf # 1 1 aimless 1 6 1.000000000 6.092064 6.09206375 # 2 1 distressed 1 6 1.000000000 6.092064 6.09206375 # 3 1 drifting 1 6 1.000000000 6.092064 6.09206375 # 4 1 movie 1 6 0.005524862

6.092064 0.03365781 # 5 1 moving 1 6 0.250000000 6.092064 1.52301594 # 6 1 slow 1 6 0.166666667 6.092064 1.01534396

###— Question 7: What is in the rows of the above partial list ###— (just comment on the first 4 columns) ###————————————————————————-

review_id: This field contains a unique identification for each movie review. Word: This column shows specific words from the film reviews. n: This column shows the frequency of each word in the review. Total: This column shows the total amount of words (excluding stop words) in the corresponding review.

matrix for analysis – needs package tm

```
require(tm)

## Loading required package: tm

## Loading required package: NLP

##
## Attaching package: 'tm'

## The following object is masked from 'package:koRpus':
##
##     readTagged

words_dtm <- words_tfidf %>%
  cast_dtm(review_id,word,n)
words_dtm

## <<DocumentTermMatrix (documents: 991, terms: 2649)>>
## Non-/sparse entries: 5412/2619747
## Sparsity           : 100%
## Maximal term length: 16
## Weighting          : term frequency (tf)

words_mat <- as.matrix(words_dtm)
dim(words_mat)

## [1]  991 2649
```

###— Question 8: What does the matrix words_mat contain?

In the matrix each row represents a unique movie review and each column represents a unique word. The values in the matrix indicate the frequency of each word in each review. The matrix has 991 rows (representing the 991 reviews) and 2649 columns (representing the 2654 unique words identified in the reviews).

select classifications for the particular 991 reviews

```
opinion <- rep(0, nrow(words_mat))
movies_response <- imdb[,c(2,3)]
rownames(movies_response) <- movies_response[,2]
```

```
for(i in 1:nrow(words_mat)) {
  opinion[i] <- movies_response[rownames(words_mat)[i],1]
}
table(opinion)
```

```
## opinion
##   0   1
## 497 494
```

We now have the opinion of the 991 movies in 'opinion'

and we have the word counts in words_mat

let's make a single regression tree

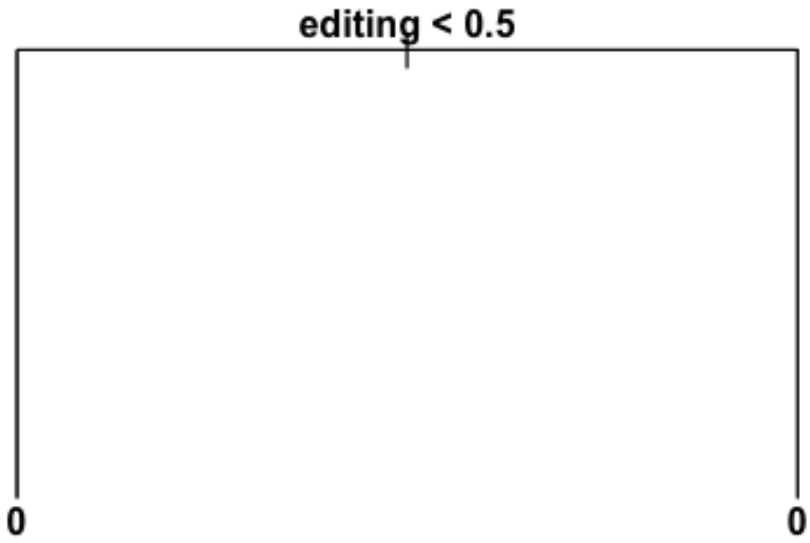
```
require(tree)
```

```
## Loading required package: tree
```

```
movie_tree <- tree(factor(opinion) ~ ., data=data.frame(words_mat))
plot(movie_tree, mar=0.1)
text(movie_tree, use.n=TRUE, font=2)
```

```
## Warning in text.default(xy$x[ind], xy$y[ind] + 0.5 * charht, rows[ind]
, :
## "use.n" is not a graphical parameter
```

```
## Warning in text.default(xy$x[leaves], xy$y[leaves] - 0.5 * charht, lab
els =
## stat, : "use.n" is not a graphical parameter
```




```
predict_opinion <- predict(movie_tree)
predict_tab <- table(predict_opinion[,2]>0.5, opinion)
predict_tab
```

```
##        opinion
##           0   1
##   FALSE 497 494
```

###— Question 9: comment on the above very bad classification tree

The root node splits based on the feature "editing < 0.5", as a result the model is using the word "editing" to decide how to classify the reviews.

Both child nodes are labeled as class 0 (negative reviews). This results that regardless of the presence or frequency of the word "editing", the model predicts all reviews as negative.

All reviews are predicted as negative 0.

The classification tree is very bad because it predicts all movie reviews as negative regardless of the actual content.

The root node's decision rule based on the word "editing" does not provide any meaningful distinction between positive and negative reviews.

The confusion matrix reveals that all positive reviews 494 are misclassified as negative, resulting in a total misclassification for positive reviews.

Lets perform random forest classification

don't split into training and test sets, just use all 991 movies

needs randomForest package

```
require(randomForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

in the following we will insert any three-digit number in as a seed

```
set.seed(106)
movies_rf <- randomForest(x=words_mat,
                          y=factor(opinion),
                          xtest=words_mat,
                          ytest=factor(opinion),
                          ntree=500, do.trace=FALSE, importance=TRUE)
```

results of random forest classification

```
movies_rf
```

```
##
## Call:
##  randomForest(x = words_mat, y = factor(opinion), xtest = words_mat,
ytest = factor(opinion), ntree = 500, importance = TRUE,      do.trace =
FALSE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 51
##
##         OOB estimate of  error rate: 25.73%
## Confusion matrix:
##     0   1 class.error
## 0 408  89   0.1790744
## 1 166 328   0.3360324
##                 Test set error rate: 4.24%
## Confusion matrix:
##     0   1 class.error
```

```
## 0 477  20  0.04024145
## 1  22 472  0.04453441
```

No. of variables tried at each split: 51

```
    OOB estimate of  error rate: 26.24%
```

Confusion matrix: 0 1 class.error 0 402 95 0.1911469 1 165 329 0.3340081 Test set error rate: 4.34% Confusion matrix: 0 1 class.error 0 476 21 0.04225352 1 22 472 0.04453441

###— Question 10: report what you get in the contents of movie_rf ###— (just comment on the "test set error rate")

The random forest model indicates strong performance with a test set error rate of 4.24%. This means that the model misclassified 4.24% of the reviews when predicting sentiment.

Confusion matrix: Class 0 (Negative reviews): 477 reviews were correctly classified as negative, while 20 reviews were misclassified as positive, resulting in a class error rate of 4.02% for class 0. Class 1 (Positive reviews): 472 reviews were correctly classified as positive, while 22 reviews were misclassified as negative, resulting in a class error rate of 4.45% for class 1.

The confusion matrix shows that the model has a low class error rate for both negative and positive reviews (4.02% and 4.45%, respectively), indicating balanced and accurate classification across both sentiment classes.