Fuzzy Logic Induced Content-Based Recommendation

EarthPleabian

Data Preparation and Processing

In this project, we use news articles from different publishers. Each article belongs to a different category: technical, entertainment, and others. This data is a subset of the news aggregator dataset from https://archive.ics.uci.edu/ml/datasets/News+Aggregator.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- 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(tidytext)
library(tm)
```

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

```
library(slam)
library(dplyr)
library(sentimentr)

set.seed(139)

cnames <- c('ID' , 'TITLE' , 'URL' , 'PUBLISHER' , 'CATEGORY' , 'STORY' , 'HOSTNAME' , 'TIMESTAMP')

data <- read_tsv('newsCorpus.csv', col_names = cnames, col_types = cols(ID = col_integer(), TITLE = col

head(data)
```

```
## # A tibble: 6 x 8
##      ID TITLE                 URL   PUBLISHER CATEGORY STORY HOSTNAME TIMESTAMP
##   <int> <chr>                 <chr> <chr>     <chr>    <chr> <chr>        <dbl>
## 1     1 Fed official says wea~ "htt~ Los Ange~ b        ddUy~ www.lat~   1.39e12
```

```
## 2     2 Fed's Charles Plosser~ "htt~ Livemint  b        ddUy~ www.liv~   1.39e12
## 3     3 US open: Stocks fall ~ "htt~ IFA Maga~ b        ddUy~ www.ifa~   1.39e12
## 4     4 Fed risks falling 'be~ "htt~ IFA Maga~ b        ddUy~ www.ifa~   1.39e12
## 5     5 Fed's Plosser: Nasty ~ "htt~ Moneynews b        ddUy~ www.mon~   1.39e12
## 6     6 Plosser: Fed May Have~ "htt~ NASDAQ    b        ddUy~ www.nas~   1.39e12
```

Every article has the following columns: * `ID`: A unique identifier
* `TITLE`: The title of the article (free text)
* `URL`: The article's URL
* `PUBLISHER`: Publisher of the article
* `CATEGORY`: Some categorization under which the articles are grouped
* `STORY`: An ID for the group of stories the article belongs to
* `HOSTNAME`: Hostname of the URL
* `TIMESTAMP`: Timestamp published

The following are some distinct publishers and categories:

```
data %>% group_by(PUBLISHER) %>% summarise()
```

```
## # A tibble: 10,938 x 1
##    PUBLISHER
##    <chr>
##  1 100.7 WZLX Classic Rock
##  2 1011now
##  3 106 JACK fm
##  4 10News
##  5 10TV
##  6 1200 WOAI
##  7 1230 WBZT
##  8 123Jump.com
##  9 123Macmini.com
## 10 12NewsNow.Com
## # i 10,928 more rows
```

```
data %>% group_by(CATEGORY) %>% summarise()
```

```
## # A tibble: 4 x 1
##   CATEGORY
##   <chr>
## 1 b
## 2 e
## 3 m
## 4 t
```

There are around 10,900 publishers and four categories. We will randomly select a sample of 5000 publishers and get the top 100 publishers by looking at the number of articles they have published:

```
data <- sample_n(data, 5000)
publisher.count <- data.frame(data %>% group_by(PUBLISHER) %>%summarise(ct =n()))
publisher.top <- head(publisher.count[order(-publisher.count$ct),],100)
head(publisher.top)
```

```
##                 PUBLISHER ct
## 1321              Reuters 47
## 1063               NASDAQ 36
## 498          Examiner.com 27
## 711       Huffington Post 27
## 1684 TheCelebrityCafe.com 25
```

```
## 486     Entertainmentwise 24
```

We can see that `Reuters` tops the list. We have retained only the articles from the top 100 publishers list for our exercise. Data frame `publisher.top` has the top 100 publishers. Now we will get the top 100 publishers, their articles, and other information.

```
data.subset <- inner_join(publisher.top, data)
```

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

```
head(data.subset)
```

```
##   PUBLISHER ct     ID
## 1   Reuters 47  36286
## 2   Reuters 47 353724
## 3   Reuters 47 192591
## 4   Reuters 47 266993
## 5   Reuters 47 322252
## 6   Reuters 47  47804
##                                                                TITLE
## 1  UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2      UPDATE 3-BMW's $1 billion plant surfs Mexican investment wave
## 3      Vietnam index plummets about 6 pct on South China sea dispute
## 4 UPDATE 1-Krispy Kreme cuts adjusted earnings outlook as costs rise
## 5                    Fed could fall behind the curve, Bullard warns
## 6   UPDATE 3-US Senate panel sets April 2 hearing on GM auto recalls
##                                                                                  URL
## 1                http://in.reuters.com/article/2014/03/23/boxoffice-idINL1N0MK05H20140323
## 2          http://in.reuters.com/article/2014/07/04/bmw-mexico-plant-idINL2N0PE1MH20140704
## 3   http://www.reuters.com/article/2014/05/08/markets-vietnam-stocks-idUSL3N0NU3EE20140508
## 4       http://in.reuters.com/article/2014/06/02/krispykreme-results-idINL3N0OJ3W320140602
## 5 http://www.reuters.com/article/2014/06/26/us-usa-fed-bullard-curve-idUSKBN0F12AI20140626
## 6       http://www.reuters.com/article/2014/03/26/gm-recall-congress-idUSL1N0MN18U20140326
##   CATEGORY                          STORY        HOSTNAME    TIMESTAMP
## 1        e d_oUk702ysXcmLMXRJe0R-XgNycCM  in.reuters.com 1.395624e+12
## 2        b deysR5eEKjCwHaMg-YORaeJFnY83M  in.reuters.com 1.404455e+12
## 3        b dNon3QbHmjev9RMZBV7iFz3EK5TRM www.reuters.com 1.399562e+12
## 4        b dVV5z8UmGUFI_-MUFIeC1yyFyYFTM  in.reuters.com 1.401799e+12
## 5        b dJela8uxiRnjVgMbiSKiq4Jvlb0cM www.reuters.com 1.403851e+12
## 6        t dRQXqhof73-73FMusfrX031zeHRDM www.reuters.com 1.395882e+12
```

```
dim(data.subset)
```

```
## [1] 1366    9
```

We join our top 100 publishers data frame `publisher.top` with data, get all the details for our top 100 publishers. Our `data.subset` has a total of 1,366 articles.

Designing the content-based recommendation engine

To begin with, we separate our data into two data frames. Then We will be using the `tm` package in R to work with our text data. Next, we do some processing of the text data.

```
title.df <- data.subset[,c('ID','TITLE')]
colnames(title.df) <- c('doc_id','text')
others.df <- data.subset[,c('ID','PUBLISHER','CATEGORY')]

library(tm)
```

```
title.reader <- DataframeSource(title.df)
corpus <- Corpus(title.reader)
readerControl=list(reader=title.reader)

getTransformations()
```

```
## [1] "removeNumbers"     "removePunctuation" "removeWords"
## [4] "stemDocument"      "stripWhitespace"
```

Calling `getTransformation` shows us the list of available functions that can be used to transform the text:

```
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeWords, stopwords("english"))
```

We remove punctuation, numbers, unnecessary white spaces, and stop words from our articles. Finally, we convert our text to lowercase. Punctuation, numbers, and whitespace may not be a good feature to distinguish one article from another. Hence, we remove them.

Finally, we proceed to build our document term matrix

```
dtm <- DocumentTermMatrix(corpus, control=list(wordlenth = c(3,10),weighting = "weightTfIdf"))
inspect(dtm[1:5,10:15])
```

```
## <<DocumentTermMatrix (documents: 5, terms: 6)>>
## Non-/sparse entries: 6/24
## Sparsity           : 80%
## Maximal term length: 10
## Sample             :
##         Terms
## Docs     bmws investment mexican plant surfs wave
##   192591    0          0       0     0     0    0
##   266993    0          0       0     0     0    0
##   322252    0          0       0     0     0    0
##   353724    1          1       1     1     1    1
##   36286     0          0       0     0     0    0
```

We use the `DocumentTermMatrix` function to create our matrix. We pass our text corpus and also pass a list to the parameter `control`. Inside the list, we say that we are interested only in words with length, so the number of characters between 3 and 10. For our cell values in our matrix, we want them to be TFIDF. Having created a document term matrix, let's create the cosine distance between the articles:

```
sim.score <- tcrossprod_simple_triplet_matrix(dtm)/(sqrt( row_sums(dtm^2) %*% t(row_sums(dtm^2)) ))
```

and we get the similarity matrix

```
sim.score[1:10,1:10]
```

```
##         Docs
## Docs         36286    353724 192591    266993    322252     47804     60270
##   36286  1.0000000 0.1250000      0 0.1178511 0.0000000 0.1250000 0.0000000
##   353724 0.1250000 1.0000000      0 0.1178511 0.0000000 0.1250000 0.0000000
##   192591 0.0000000 0.0000000      1 0.0000000 0.0000000 0.0000000 0.0000000
##   266993 0.1178511 0.1178511      0 1.0000000 0.0000000 0.1178511 0.1360828
##   322252 0.0000000 0.0000000      0 0.0000000 1.0000000 0.0000000 0.0000000
##   47804  0.1250000 0.1250000      0 0.1178511 0.0000000 1.0000000 0.0000000
```

```
##   60270  0.0000000 0.0000000      0 0.1360828 0.0000000 0.0000000 1.0000000
##   398725 0.1250000 0.2500000      0 0.1178511 0.1443376 0.1250000 0.0000000
##   101350 0.0000000 0.0000000      0 0.0000000 0.0000000 0.1581139 0.0000000
##   203598 0.0000000 0.0000000      0 0.0000000 0.0000000 0.0000000 0.0000000
##         Docs
## Docs        398725    101350 203598
##   36286  0.1250000 0.0000000      0
##   353724 0.2500000 0.0000000      0
##   192591 0.0000000 0.0000000      0
##   266993 0.1178511 0.0000000      0
##   322252 0.1443376 0.0000000      0
##   47804  0.1250000 0.1581139      0
##   60270  0.0000000 0.0000000      0
##   398725 1.0000000 0.0000000      0
##   101350 0.0000000 1.0000000      0
##   203598 0.0000000 0.0000000      1
```

Searching

Having created our similarity matrix, we can leverage that matrix to find a match for any given document. We will be using the `sim.score` created in the previous step to perform the search. Let's say we want to find similar articles to article `36286`:

```
match.docs <- sim.score["36286",]
match.docs
```

```
##     36286    353724    192591    266993    322252     47804     60270    398725
## 1.0000000 0.1250000 0.0000000 0.1178511 0.0000000 0.1250000 0.0000000 0.1250000
##    101350    203598      4676    181385    212332    163646     51791    267770
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1336306 0.0000000 0.0000000
##    310399    260838    327205    195962    306435    230566    155784    401470
## 0.1178511 0.0000000 0.0000000 0.0000000 0.1443376 0.1250000 0.0000000 0.0000000
##    234668    402744    302682    156135    307682    377401    371053    413420
## 0.0000000 0.1250000 0.1118034 0.0000000 0.0000000 0.0000000 0.1250000 0.0000000
##    255452    304880    277559     92168    294174    179751    247177     77152
## 0.1250000 0.1118034 0.0000000 0.1443376 0.1250000 0.1336306 0.0000000 0.0000000
##    343254    317522    117017      1440    192128    165609     96179    357355
## 0.0000000 0.0000000 0.1250000 0.1178511 0.1178511 0.0000000 0.1250000 0.0000000
##    334251    197210      8689    391558    144791    150378    371743    259180
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1336306 0.0000000 0.0000000
##    293817    213014    302377    188553    307753    128387    354263     30940
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    200683    204250    393295     95977    199974    171384    315231    155957
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    196395    176240    192149    181085    330375    308077    339457    410418
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    149616     88391    217785    275619    333208    126701      7289     66811
## 0.1250000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    118917    287724    341328    380944     39704     72120     92001    182437
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    381531     21531    164925    150645    140107     34105     21532     45795
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    249881      7247    343026    179101     26222     15463    215637     15793
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    143478    358435      2805    234933    169010     55444    367332    237801
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     29214     58952    118468    116696    127716    196785    309513    352516
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     93113    183175    211096    271961    390485    274149    115916    421153
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     46289    300463      6458     19905    157486     67767    135109     23001
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    365416    323787    145232    376350    329090    123120    254885    313386
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    416731    168522    269848    179348      2926     14180    106978    126912
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    258230    390092    320337    280235    337624    258125    293026    142263
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##      6659    265582    161411    251889     86210     93511    258085    362544
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    303676    154698     49737     49410    363514    342649    320844    304402
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    325201    270579    135434    362669    397149     15045    254434    358450
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    274766     54316    293103    127006     14564    332209     58912    402817
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1767767 0.0000000
##    408330     67852    151443    374057    313608     37038     75348    375866
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    276770    191370    364918    223206    299352    397607    106483    265706
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    328739    354568    247391    412065    420483    148063    363403    178302
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    404181    422693    279147     75771     15682    317940    317769    248448
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     77254     92154    305682    117406     57043    185641     13178    250416
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    230511      3355    371871    201409     38698     86813     83712    173843
## 0.0000000 0.0000000 0.0000000 0.0000000 0.1250000 0.0000000 0.0000000 0.0000000
##    298336    230702    405397     50521    217693     13910    357327    188266
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    180838     50423     60092    346145    231787    348317    393266    112810
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     31304    378704    173698    260950    313179    369128      2670    380732
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     31766    387068    273367    124890    402875    162458    186011    267941
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    229034    410174    359562    242699    102258    245913     36925    406600
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    222098    257312    270425    211133    205716    240859    134104    368829
## 0.0000000 0.0000000 0.0000000 0.0000000 0.1178511 0.0000000 0.0000000 0.0000000
##    302351    172110    180482    353317    317738    160768     31344    204996
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     77518    239803     17804    302216    364508    165889    366573    306556
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    302129    156230    104689    386499    241113    226329    152064    149234
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     45370     55983    237045       644    410808    160366    391439    108106
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    321339     37867     61074    231758    237679    413071    116840    348278
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    226940    343194    398599    255510    184474    192631     35474    261404
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     81028    168189     33528    284850    292557    259307     10282    342483
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    292558    355045    321921    374019    364136    292794    374012     22939
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    275761    295701     72812    140283    187672    312839    216010    138700
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    292103      3479    229299     14993    376299     23498    317573    363504
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    258487    222126    313749    360374    185423    343429    336860     21613
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    403757    277316     31792    327355    385933    286194     68787    128258
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    296767    224292    313613     16247     89849    100919    415584    126459
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    265252    199801    144199      4888    202859    401332     37477    217075
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    369718     46995    222881    159980    293010    124782     35569    147420
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    264246     71833    159574    306755    406855    203949    237923    286797
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     46970     68584    246699    413303    217618    181387    269296    329772
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    419780    378930    275597    170538    413829    109999    278898    347939
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    322113    270497    140112    133483    153647    106247    243036    313779
## 0.0000000 0.0000000 0.0000000 0.1443376 0.0000000 0.0000000 0.0000000 0.0000000
##    244261    296565    206208    333498     34095    130311     78785    335792
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    190644    125876    215998    285910    293158    206842     47658    139824
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    318848    229996    371120    317244    157407    264617    410829    376052
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    159884    166696    420968    284951    345652    100368    173609    101833
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    389679     56120    101327    409952    338917    249874    156190    197217
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    127901    420932    371727    176603     43446     55719    175196    251100
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    130559     67293    370120    342615    357849    175870    334879    106187
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     54046    240535     53254    248984    104955    414529    412710    279302
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    156234    371659     33366    163191    238836    217163    155917     68234
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    297449    212790     95804     34979    213169    237950    103208    103745
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    361126    354848     39753     19786     98434    202955    366575    420966
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    118776    347546    178633     10892     10901      3624    299168    173167
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    265294    225816     82822    386398    203540    413509    345981     78222
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    203280    204872    308448    209289    263234     40794      1492    326556
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##      3465    315356    290055     39931    124167    100564    236918    362423
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    142933    219003    211943    198580    177283    296806    339551    237830
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##      8697    405941     57228     31230    414212    192558    212845    144962
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    339538     87334     70533    353605    171331     35465    137091     61003
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    185581    124401    405513    383484    245191    405334    310617    315652
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    290779    387037     49838    390201    124776    394695     55313    269521
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##       380    244311    213971    142575    285966    262762    152303    132663
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    154492    213703    112937    280595    283448      1530    391310      1306
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    322682    170542    370939    189479     98121     34807     81339    232196
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    343341    159869     83600     10047    289950    236492    394847    404337
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##      6406    202896    240996     15724    291120    152446    387177    396013
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    147454    127447    340356    184189    200650    418086    321602     56274
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     78709    230722    360712    392575    312474    266030    157453    379588
## 0.0000000 0.0000000 0.0000000 0.0000000 0.2132007 0.0000000 0.0000000 0.0000000
##    143601    291527     44135    157902    143538    414077    278403     82470
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     23073    270913    354703    303086    349080    384139     66112    324274
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    128770    145138    257078    223802    157540    120888    217096    331938
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    298867    410592    171367    312216    235202    171406    255050    255122
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    144720    255492      1193    139376    208703      5504    180647    204742
## 0.0000000 0.0000000 0.1443376 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     48662    272504    204545    225871    181139     87668    387090    197273
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    255751    125754    359917     60272    105050    136788    156576    350135
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     24293    325390    238757    192262     83380    406318    361640    259366
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    336216    204458    145339    322921    120897     85268    308667    402184
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    173560     80493    276250    412575    173854    307586    276884    229824
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    385724    224306    368577     56657    252595    191432     21670    284020
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    415312    253282    343318     65016     45419     88503     57596    174379
## 0.0000000 0.1250000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     17836    298952     57362     10833    291260    205575    160177    339895
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     83986    389621    219082    287466    168549    202204     49140     73148
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    300260    236901     42604     89949    140527      7683    268113    172478
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    356837    125115    159201    226107    310587    330456     38041    276211
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     97172    150070    160592     21735     94885    235901    316448     26065
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     29268    277353      2319    419850    189303    269063    137454    323167
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    347904    169998    420514    220295    227500    264163    419753    258490
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     23587     40083    409544     42817    242443     34028    142363    408037
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     26130     20360    405260    175675    361243    421572    415059     17873
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    389072     85624     10941    202456    403558    376315    168381    215689
## 0.0000000 0.0000000 0.0000000 0.0000000 0.2357023 0.0000000 0.0000000 0.0000000
##    196302    366681     27335    130471    149046    118852    261415    390622
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    153030    224171    208017    372552     12332    272690    209411    366133
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     11125    208808    377539    155664    356064    307677    146602    357290
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    228074     77327    157787    179199    334393    268545    417975     39489
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    106275    115133    215697    141520    217889    131388    290803    190480
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    310180    258725     91726    132331     90978    311411    330155    313951
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     76141    352258    353600     79814     91683     80237     77849    305779
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    255648    202634    276889    344679    279352    129859    233395    253167
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    144151    369809    158829      6792    376619    238937     19032    407443
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     19951    109331    130158    415100    178285    361357    206418    222708
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    287952    386409    177786     39950    258163    381468    339617     48101
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    325458    409802     27830    181988     78327    265118    355170    213508
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    238892    373996    271596     72220    243801    167625      3764    270692
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    342424     86742    218425    338402    112600     95736    311326     80269
## 0.0000000 0.3162278 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    371019    247955    383418    155821    263165    139603    366985    105467
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    412637    301848     92323    394076    177341     85209    207781    119106
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    163919    416802    209635    404787    316493    399274    141757    316316
## 0.0000000 0.0000000 0.0000000 0.0000000 0.1336306 0.1178511 0.0000000 0.0000000
##    166329    361854    213458    344191     56467    319770     33064    333003
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    169551     58966    172819    178754    236869     11816    295684    328953
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    417626     36230    187571    283973    281016    263590     20706    168823
## 0.0000000 0.4743416 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     30877     45094     33015    303542    156605    170968    172675    167436
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    125987    229482    273590    312007    394680      5542    188992    310868
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    291085    128403    181063    120575    212597     83173    410027     51151
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     46705     20847    262900    174666     97169    415789    260396    303538
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    269876    275767     71815    149627     59912     17853     60439    417311
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    207611    342879     59613    238187    420101    399537    191471    118542
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    270753    325103    380345    346581     10894    355760    243568    403905
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.2500000 0.0000000
##    396782    351447    317564    401205     99889    418724     32424    112337
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    313631    322073    363587    388900     25511    168764    135285    218027
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    249083    363888    228978    184406    346711     92433    118174    191187
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    101696    243660     91470    111803    403229    404835    148821    417492
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    333044     71239     87654    338761    173358    196346    357042    180721
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     57063    334527     68368    403520    274830     15190      6587    191128
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    255010    148407    110902    271762    153540    144041    366083     23618
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    358552    372833    396056    328550    411528    269768    274604     93757
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    361034    103667    242758    168063    252135    332337     67739     74467
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    322048     23296    247453      6469    320379     54074    179176      3192
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    131343     19773    337448     54652    247393     75720    147898     10986
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    221992    284597    271166    271661    295604    401779    175439    325047
## 0.0000000 0.0000000 0.0000000 0.1118034 0.0000000 0.0000000 0.0000000 0.0000000
##    137472    385519    286403    291992    207977    209034    185405     30290
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    328823    103019    262386    375039     32458    304546    110565    421290
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    140843    260210    322634     51045    412970    403604     29984     44536
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     66964    119341     13317    371287     31723    356079    356178     53407
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    135579     93760    150634    365190     27589    218844     21005    374767
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.2886751 0.0000000 0.0000000
##    121026    278823     77587    246953    220173    365776    260248    343265
```

```
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    260921    202879    313026    206495    140992    124232    295608    222812
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    243134    342656     10843    119342     23345     44035    138752     13055
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     21901    404175    174252    118402    124129    247214    320876    302858
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    369576    396682      6127     78810    189716    258483    287779    328846
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     11642    369608     43390    285383    324759     65757    119037    300308
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    206525     78039    271093    190382    254143    252058    351503     72370
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     58880    165374    142282     40527    402862    168972      6487    407874
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    137338    162372    309287     17831    335762    392896    217729    278009
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    388226     38382     57818    338710       836     44109    334466    314743
## 0.0000000 0.0000000 0.1250000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    230968       621    330332    255127     14634    365257    416106    175312
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    361136    378786    358099    300309    254469    357441    172826    149974
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     75190    409833    248674     29212     13920    147279    250491    252745
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    250072    406462    245755    334532     90835     96046     29959    246849
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    214423    421650    325501    373602    238351    382488     96399    220980
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    119049    389695    247337    380867    261001    403992     53541    416145
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    261715    201194    378891    129301    279137    117683     66618    347047
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    106217    167425    301262     84499    205130    177153     15432     61365
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    211689     71728    248440    348636    349409    383920    208382     87680
## 0.0000000 0.1250000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##    234637    376773    339251    208467    269596    368034     10176    412267
## 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     88428     52939    155675     60080    231558    408627    147748     27055
## 0.0000000 0.1250000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##     78962    228642     83033    202276    258094    144183
## 0.0000000 0.0000000 0.0000000 0.2500000 0.2500000 0.0000000
```

We go to our `match.doc` similarity matrix and pick up row `36286`. Now, this row has all the other articles and their similarity scores. Let's now take this row and make a data frame:

```
match.df <- data.frame(ID = names(match.docs), cosine = match.docs, stringsAsFactors=FALSE)
match.df$ID <- as.integer(match.df$ID)
head(match.df)
```

```
##            ID    cosine
## 36286   36286 1.0000000
## 353724 353724 0.1250000
## 192591 192591 0.0000000
```

```
## 266993 266993 0.1178511
## 322252 322252 0.0000000
## 47804   47804 0.1250000
```

Our `match.df` data frame now contains all the matching documents for `36286`. Now, we are going to recommend only the top 30 matches:

```
match.refined<-head(match.df[order(-match.df$cosine),],30)
head(match.refined)
```

```
##            ID    cosine
## 36286   36286 1.0000000
## 36230   36230 0.4743416
## 86742   86742 0.3162278
## 218844 218844 0.2886751
## 243568 243568 0.2500000
## 202276 202276 0.2500000
```

Now that we have the matching documents, we need to present them in a ranked order. In order to rank the results, we are going to calculate some additional measures and use fuzzy logic to get the final ranking score. Before we go ahead and calculate the additional measures, let's merge `title.df` and `other.df` with `match.refined`:

```
colnames(title.df) <- c('ID','TITLE')
match.refined <- inner_join(match.refined, title.df)
```

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

```
match.refined <- inner_join(match.refined, others.df)
```

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

```
head(match.refined)
```

```
##       ID    cosine
## 1  36286 1.0000000
## 2  36230 0.4743416
## 3  86742 0.3162278
## 4 218844 0.2886751
## 5 243568 0.2500000
## 6 202276 0.2500000
##                                                                                TITLE
## 1              UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2                                              'Divergent' tops weekend box office
## 3                                              Captain America's box office record
## 4                           'Godzilla' flattens box office with monstrous debut
## 5                      X Men Days of Future Past Box Office: Millions For Mutants!
## 6 'Neighbors' Crashing Spider-Man's Party at Box Office with $45 Million Opening
##              PUBLISHER CATEGORY
## 1              Reuters        e
## 2              Newsday        e
## 3    Belfast Telegraph        e
## 4   Detroit Free Press        e
## 5 The Hollywood Gossip        e
## 6              Variety        e
```

Polarity scores

We are going to leverage the `sentimentr` R package to learn the sentiments of the articles we have collected. Let's look at how to score using the `sentiment` function :

```
sentiment.score <- sentiment(match.refined$TITLE)
head(sentiment.score)
```

```
##    element_id sentence_id word_count  sentiment
## 1:          1           1          9 -0.2666667
## 2:          2           1          5  0.4472136
## 3:          3           1          5  0.0000000
## 4:          4           1          7 -0.1889822
## 5:          5           1         11  0.0000000
## 6:          6           1         11 -0.3015113
```

The `sentiment` function in `sentimentr` calculates a score between -1 and 1 for each of the articles. If a text has multiple sentences, it will calculate the score for each sentence. A score of -1 indicates that the sentence has a very negative polarity. A score of 1 means that the sentence is very positive. A score of 0 refers to the neutral nature of the sentence.

However, we need the score at an article level and not at a sentence level, so we can take an average value of the score across all the sentences in a text.

```
sentiment.score <- sentiment.score %>% group_by(element_id) %>% summarise(sentiment = mean(sentiment))
head(sentiment.score)
```

```
## # A tibble: 6 x 2
##   element_id sentiment
##        <int>     <dbl>
## 1          1    -0.267
## 2          2     0.447
## 3          3     0
## 4          4    -0.189
## 5          5     0
## 6          6    -0.302
```

Here, the `element_id` refers to the individual article. By grouping `element_id` and calculating the average, we can get the sentiment score at an article level. We now have the scores for each article. Next we update the `match.refined` data frame with the polarity scores

```
match.refined$polarity <- sentiment.score$sentiment
head(match.refined)
```

```
##       ID    cosine
## 1  36286 1.0000000
## 2  36230 0.4743416
## 3  86742 0.3162278
## 4 218844 0.2886751
## 5 243568 0.2500000
## 6 202276 0.2500000
##                                                                          TITLE
## 1              UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2                                           'Divergent' tops weekend box office
## 3                                           Captain America's box office record
## 4                              'Godzilla' flattens box office with monstrous debut
## 5                      X Men Days of Future Past Box Office: Millions For Mutants!
## 6 'Neighbors' Crashing Spider-Man's Party at Box Office with $45 Million Opening
```

```
##              PUBLISHER CATEGORY   polarity
## 1              Reuters        e -0.2666667
## 2              Newsday        e  0.4472136
## 3    Belfast Telegraph        e  0.0000000
## 4   Detroit Free Press        e -0.1889822
## 5 The Hollywood Gossip        e  0.0000000
## 6              Variety        e -0.3015113
```

Jaccard's distance

While ranking the matched articles, we want to also include the category and publisher columns. Let's proceed to include those columns:

```
target.publisher <- match.refined[1,]$PUBLISHER
target.category <- match.refined[1,]$CATEGORY
target.polarity <- match.refined[1,]$polarity
target.title <- match.refined[1,]$TITLE
```

We need the publisher, category, and the sentiment details of the document we are searching for. Fortunately, the first row of our `match.refined` data frame stores all the details related to `36286`. For the rest of the articles, we need to find out if they match the publisher and category of document `36286`.

```
match.refined$is.publisher <- match.refined$PUBLISHER == target.publisher
match.refined$is.publisher <- as.numeric(match.refined$is.publisher)
match.refined$is.category <- match.refined$CATEGORY == target.category
match.refined$is.category <- as.numeric(match.refined$is.category)
```

With the two new columns, we can calculate the Jaccard's distance between document `36286` and all the other documents in the `match.refined` data frame.

```
match.refined$jaccard <- (match.refined$is.publisher + match.refined$is.category)/2
```

The Jaccard index measures the similarity between two sets, and is a ratio of the size of the intersection and the size of the union of the participating sets. Here we have only have two elements, one for publisher and one for category, so our union is 2. The numerator, by adding the two Boolean variable, we get the intersection.

Finally, we also calculate the absolute difference (Manhattan distance) in the polarity values between the articles in the search results and our search article. We do a min/max normalization of the difference score.

```
match.refined$polaritydiff <- abs(target.polarity - match.refined$polarity)

range01 <- function(x){(x-min(x))/(max(x)-min(x))}
match.refined$polaritydiff <- range01(unlist(match.refined$polaritydiff))
head(match.refined)
```

```
##       ID    cosine
## 1  36286 1.0000000
## 2  36230 0.4743416
## 3  86742 0.3162278
## 4 218844 0.2886751
## 5 243568 0.2500000
## 6 202276 0.2500000
##                                                                    TITLE
## 1             UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2                                          'Divergent' tops weekend box office
## 3                                          Captain America's box office record
## 4                          'Godzilla' flattens box office with monstrous debut
## 5                  X Men Days of Future Past Box Office: Millions For Mutants!
```

```
## 6 'Neighbors' Crashing Spider-Man's Party at Box Office with $45 Million Opening
##              PUBLISHER CATEGORY   polarity is.publisher is.category jaccard
## 1              Reuters        e -0.2666667            1           1     1.0
## 2              Newsday        e  0.4472136            0           1     0.5
## 3    Belfast Telegraph        e  0.0000000            0           1     0.5
## 4   Detroit Free Press        e -0.1889822            0           1     0.5
## 5 The Hollywood Gossip        e  0.0000000            0           1     0.5
## 6              Variety        e -0.3015113            0           1     0.5
##   polaritydiff
## 1   0.00000000
## 2   0.83159146
## 3   0.31063714
## 4   0.09049376
## 5   0.31063714
## 6   0.04059019
```

We remove some of the unwanted fields from the `match.refined` data frame. Finally, we have the ID, cosine distance, title, publisher, category, Jaccard score, and the polarity difference.

```
match.refined$is.publisher = NULL
match.refined$is.category = NULL
match.refined$polarity = NULL
match.refined$sentiment = NULL
head(match.refined)
```

```
##       ID    cosine
## 1  36286 1.0000000
## 2  36230 0.4743416
## 3  86742 0.3162278
## 4 218844 0.2886751
## 5 243568 0.2500000
## 6 202276 0.2500000
##                                                                          TITLE
## 1              UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2                                          'Divergent' tops weekend box office
## 3                                          Captain America's box office record
## 4                           'Godzilla' flattens box office with monstrous debut
## 5                     X Men Days of Future Past Box Office: Millions For Mutants!
## 6 'Neighbors' Crashing Spider-Man's Party at Box Office with $45 Million Opening
##              PUBLISHER CATEGORY jaccard polaritydiff
## 1              Reuters        e     1.0   0.00000000
## 2              Newsday        e     0.5   0.83159146
## 3    Belfast Telegraph        e     0.5   0.31063714
## 4   Detroit Free Press        e     0.5   0.09049376
## 5 The Hollywood Gossip        e     0.5   0.31063714
## 6              Variety        e     0.5   0.04059019
```

Ranking search results

We need to perform our ranking based on the following metrics we have calculated: * Cosine similarity * Jaccard index * Polarity difference In this project, we will leverage fuzzy logic programming to do the search result ranking. We will be using the `sets` R package for our fuzzy logic programming:

```
library(sets)
```

```
##
```

```
## Attaching package: 'sets'

## The following objects are masked from 'package:lubridate':
##
##     as.interval, interval, is.interval

## The following object is masked from 'package:forcats':
##
##     %>%

## The following object is masked from 'package:stringr':
##
##     %>%

## The following object is masked from 'package:dplyr':
##
##     %>%

## The following object is masked from 'package:purrr':
##
##     %>%

## The following object is masked from 'package:tidyr':
##
##     %>%

## The following object is masked from 'package:tibble':
##
##     %>%
sets_options("universe", seq(from = 0, to = 1, by = 0.1))
```

The first step is to set up our universe. We define the range of values and the granularity of the values we will be dealing with in our universe. Our cosine, Jaccard, and polarity are all normalized to have a value between zero and one. Hence, the range of our universe is set between zero and one.

The first step in fuzzy logic programming is to define the linguistic variables we will be dealing with:

```
variables <- set(
  cosine = fuzzy_partition(varnames = c(vlow = 0.2, low = 0.4, medium = 0.6, high = 0.8), FUN = fuzzy_c
  jaccard = fuzzy_partition(varnames = c(close = 1.0, halfway = 0.5, far = 0.0), FUN = fuzzy_cone , radi
  polarity = fuzzy_partition(varnames = c(same = 0.0, similar = 0.3,close = 0.5, away = 0.7), FUN = fuzz
  ranking = fuzzy_partition(varnames = c(H = 1.0, MED = 0.7 , M = 0.5, L = 0.3), FUN = fuzzy_cone , radi
```

For each variable, we define the various linguistic values and the `fuzzy membership` function. For example, for our linguistic variable `cosine`, the linguistic values include `vlow, low, medium, and high`.

Based on the interaction between the linguistic variables cosine, jaccard, and polarity, the ranking linguistic variables are assigned different linguistic values. These interactions are defined as rules. Having defined the linguistic variables, the linguistic values, and the membership function, we proceed to write down our fuzzy rules:

```
rules <- set(
######### Low Ranking Rules ###################
fuzzy_rule(cosine %is% vlow, ranking %is% L),
fuzzy_rule(cosine %is% low || jaccard %is% far || polarity %is% away, ranking %is% L),
fuzzy_rule(cosine %is% low || jaccard %is% halfway || polarity %is% away, ranking %is% L),
fuzzy_rule(cosine %is% low || jaccard %is% halfway || polarity %is% close, ranking %is% L),
fuzzy_rule(cosine %is% low || jaccard %is% halfway || polarity %is% similar, ranking %is% L),
fuzzy_rule(cosine %is% low || jaccard %is% halfway || polarity %is% same, ranking %is% L),
```

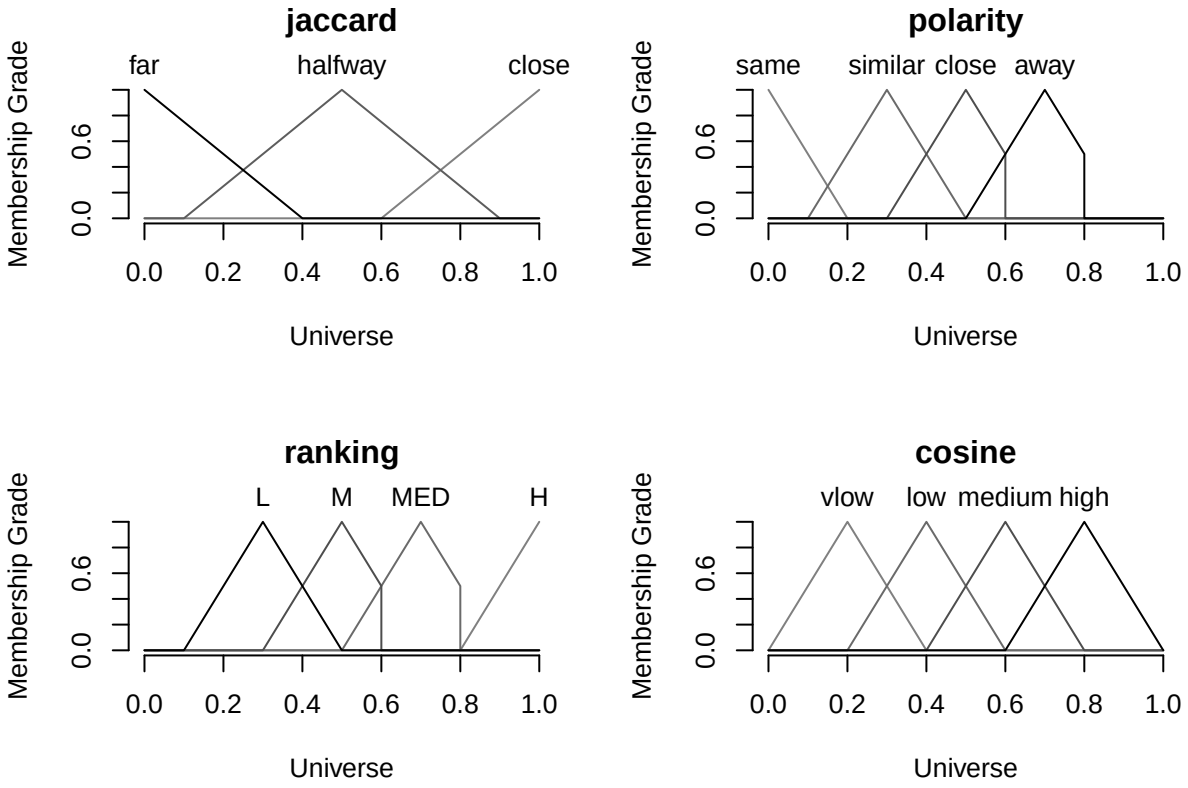
```
fuzzy_rule(cosine %is% medium || jaccard %is% far || polarity %is% away, ranking %is% L),
################ Medium Ranking Rules ##################
fuzzy_rule(cosine %is% low || jaccard %is% close|| polarity %is% same,ranking %is% M),
fuzzy_rule(cosine %is% low && jaccard %is% close && polarity %is% similar, ranking %is% M),
################ Median Ranking Rule ##################
fuzzy_rule(cosine %is% medium && jaccard %is% close && polarity %is% same, ranking %is% MED),
fuzzy_rule(cosine %is% medium && jaccard %is% halfway && polarity %is% same, ranking %is% MED),
fuzzy_rule(cosine %is% medium && jaccard %is% close && polarity %is% similar, ranking %is% MED),
fuzzy_rule(cosine %is% medium && jaccard %is% halfway && polarity %is% similar, ranking %is% MED),
############### High Ranking Rule ####################
fuzzy_rule(cosine %is% high,ranking %is% H))
```

With the `rules` and `linguistic variables` defined, we can now put our complete fuzzy system together:

```
ranking.system <- fuzzy_system(variables, rules)
print(ranking.system)
```

```
## A fuzzy system consisting of 4 variables and 14 rules.
##
## Variables:
##
## jaccard(close, halfway, far)
## polarity(same, similar, close, away)
## ranking(H, MED, M, L)
## cosine(vlow, low, medium, high)
##
## Rules:
##
## cosine %is% low && jaccard %is% close && polarity %is% similar => ranking %is% M
## cosine %is% medium && jaccard %is% close && polarity %is% same => ranking %is% MED
## cosine %is% medium && jaccard %is% close && polarity %is% similar => ranking %is% MED
## cosine %is% medium && jaccard %is% halfway && polarity %is% same => ranking %is% MED
## cosine %is% medium && jaccard %is% halfway && polarity %is% similar => ranking %is% MED
## cosine %is% low || jaccard %is% far || polarity %is% away => ranking %is% L
## cosine %is% low || jaccard %is% close || polarity %is% same => ranking %is% M
## cosine %is% low || jaccard %is% halfway || polarity %is% away => ranking %is% L
## cosine %is% low || jaccard %is% halfway || polarity %is% same => ranking %is% L
## cosine %is% low || jaccard %is% halfway || polarity %is% close => ranking %is% L
## cosine %is% low || jaccard %is% halfway || polarity %is% similar => ranking %is% L
## cosine %is% medium || jaccard %is% far || polarity %is% away => ranking %is% L
## cosine %is% high => ranking %is% H
## cosine %is% vlow => ranking %is% L
```

```
plot(ranking.system)
```



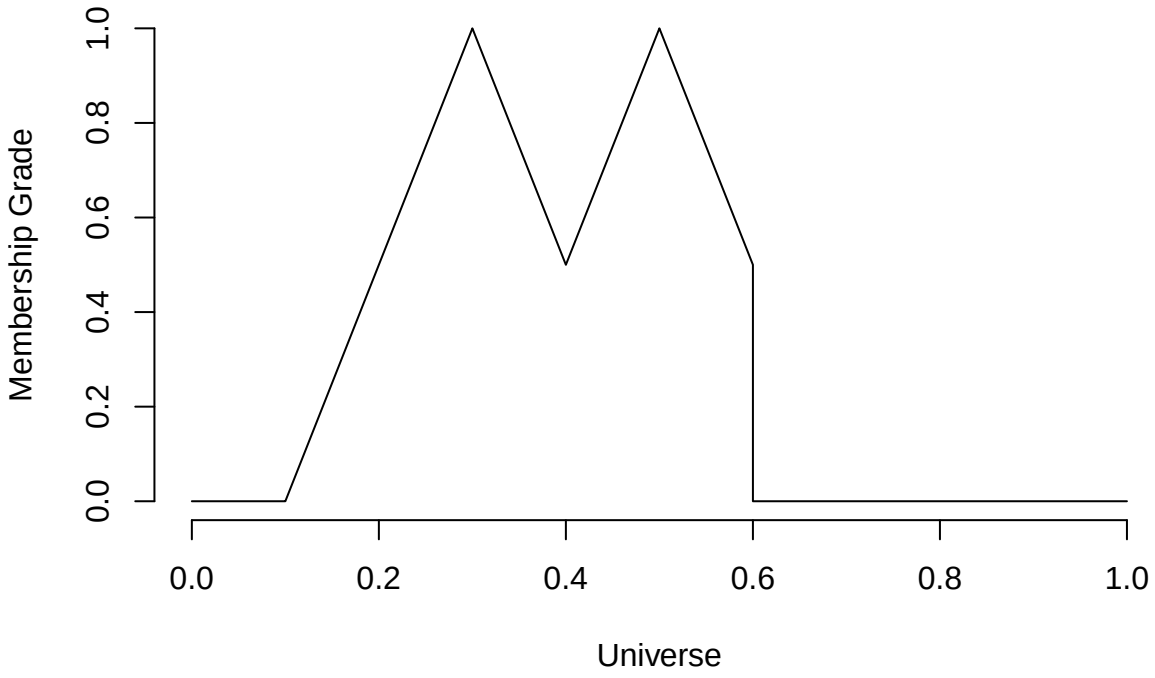
The final plot reveals the fuzziness in the boundary for different linguistic variables. Compare this with a hard-coded `if-else` logic system.

We can now proceed to use this system to do the ranking. Let's do the ranking on a single example:

```
fi <- fuzzy_inference(ranking.system, list(cosine = 0.5000000, jaccard = 0, polarity=0.00000000))
gset_defuzzify(fi, "centroid")
```

```
## [1] 0.4
```

```
plot(fi)
```



For given values of cosine, polarity, and Jaccard, we get a ranking score of 0.4. Now we can use this score to rank the results.

Let's generate the rankings for all the articles in `match.refined`:

```
get.ranks <- function(dataframe){
  cosine = as.numeric(dataframe['cosine'])
  jaccard = as.numeric(dataframe['jaccard'])
  polarity = as.numeric(dataframe['polaritydiff'])
  fi <- fuzzy_inference(ranking.system, list(cosine = cosine, jaccard = jaccard, polarity=polarity))
  return(gset_defuzzify(fi, "centroid"))
}

match.refined$ranking <- apply(match.refined, 1, get.ranks)
match.refined <- match.refined[order(-match.refined$ranking),]
head(match.refined)
```

```
##       ID    cosine
## 1  36286 1.0000000
## 2  36230 0.4743416
## 3  86742 0.3162278
## 4 218844 0.2886751
## 5 243568 0.2500000
## 6 202276 0.2500000
##                                                                      TITLE
## 1             UPDATE 1-'Divergent' teen warriors defeat 'Muppets' at box office
## 2                                          'Divergent' tops weekend box office
## 3                                          Captain America's box office record
```

```
## 4                            'Godzilla' flattens box office with monstrous debut
## 5                     X Men Days of Future Past Box Office: Millions For Mutants!
## 6 'Neighbors' Crashing Spider-Man's Party at Box Office with $45 Million Opening
##              PUBLISHER CATEGORY jaccard polaritydiff ranking
## 1              Reuters        e     1.0   0.00000000     0.4
## 2              Newsday        e     0.5   0.83159146     0.3
## 3    Belfast Telegraph        e     0.5   0.31063714     0.3
## 4   Detroit Free Press        e     0.5   0.09049376     0.3
## 5 The Hollywood Gossip        e     0.5   0.31063714     0.3
## 6              Variety        e     0.5   0.04059019     0.3
```

The `get.ranks` function is applied in each row of `match.refined` to get the fuzzy ranking. Finally, we sort the results using this ranking.