Fuzzy Logic Induced Content-Based Recommendation

Nguyen Quy Khoi

June 01, 2023

Contents

1 Introduction to content-based recommendation

When a friend comes to you for a movie recommendation, you don't arbitrarily start shooting movie names. You try to suggest movies while keeping in mind your friend's tastes. Content-based recommendation systems, also known as content-based filtering methods, try to mimic the exact same process.

Consider a scenario in which a user is browsing through a list of products. Given a set of products and the associated product properties, when a person views a particular product, content-based recommendation systems can generate a subset of products with similar properties to the one currently being viewed by the user.

The following example demonstrates how content-based recommendation systems work.

1.1 Introduction to wine suggestion system

An example of using content-based recommendation is wine suggestion. We will use the wine data set from UCI Machine Learning Repository. This data set is the result of the chemical analysis of wine grown in the same region in Italy. We have data from three different cultivars (from an assemblage of plants selected for desirable characters).

In the data set, rows represent each wine brand, while the columns represent the properties of the wines. The table is extracted from UCI machine learning repository as follows:

```
# data.table provides a high-performance version of base R's data.frame with
# syntax and feature enhancements for ease of use, convenience and
# programming speed.
library(data.table)
library(here); library(skimr); library(janitor)
# differences between "<-" and "="
# "<-" is for assignment, standard everywhere
# "=" is for passing arguments, (allowed only for top level env)
#wine.data <- fread('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wi
wine.data <- fread('data\\wine.data')
skim_without_charts(as.data.frame(wine.data))
```

Table 1: Data summary

Name	as.data.frame(wine.data)
Number of rows	178
Number of columns	14
Column type frequency:	
numeric	14
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
V1	0	1	1.94	0.78	1.00	1.00	2.00	3.00	3.00
V2	0	1	13.00	0.81	11.03	12.36	13.05	13.68	14.83
V3	0	1	2.34	1.12	0.74	1.60	1.87	3.08	5.80
V4	0	1	2.37	0.27	1.36	2.21	2.36	2.56	3.23
V5	0	1	19.49	3.34	10.60	17.20	19.50	21.50	30.00
V6	0	1	99.74	14.28	70.00	88.00	98.00	107.00	162.00
V7	0	1	2.30	0.63	0.98	1.74	2.36	2.80	3.88
V8	0	1	2.03	1.00	0.34	1.20	2.13	2.88	5.08
V9	0	1	0.36	0.12	0.13	0.27	0.34	0.44	0.66
V10	0	1	1.59	0.57	0.41	1.25	1.56	1.95	3.58
V11	0	1	5.06	2.32	1.28	3.22	4.69	6.20	13.00
V12	0	1	0.96	0.23	0.48	0.78	0.96	1.12	1.71
V13	0	1	2.61	0.71	1.27	1.94	2.78	3.17	4.00
V14	0	1	746.89	314.91	278.00	500.50	673.50	985.00	1680.00

1.2 Chemical properties similarity between two wines

We have a total of 14 columns. The first column `V1` represents the cultivar, which is assigned in `wine.type`. The remaining columns are the chemical properties of the wine, which are stored in `wine.features`:

```r
wine.type <- wine.data[,1]
wine.features <- wine.data[,-1]
# scale feature properties such that each column has mean=0 and std=1
wine.features.scaled <- scale(wine.features)
# convert feature properties into matrix to perform linear algebra operations
wine.mat <- data.matrix(wine.features.scaled)
# assign row names for matrix wine.mat (name = sequence from 1 to 178)
rownames(wine.mat) <- seq(1:dim(wine.features.scaled)[1])
# transpose the matrix, feature properties are row, wines are column
# to perform pearson coefficient calculation between 2 columns
wine.mat <- t(wine.mat)
```

The pairwise correlation in the numerical matrix is measured using Pearson coefficient after scaling. In this problem, the Pearson coefficient matrix returns *the similarity between 2 wines* (i.e., compare 2 columns).

The output is the similarity matrix, which shows how closely related items are. The values range from -1 for perfect negative correlation, when two items have attributes that move in opposite directions, and +1 for perfect positive correlation, when attributes for the two items move in the same direction. The diagonal values will be +1, as we are comparing a wine to itself.

```r
# perform pearson correlation calculation, which returns an 178x178 matrix
cor.matrix <- cor(wine.mat, use="pairwise.complete.obs", method="pearson")
```

1.3 Test case: Find top 4 similar wines to wine 3

In this problem, the similarity matrix is useful for finding similar wines (target column) in the wine's catalog. For example, if a customer likes wine 3 and wants to know if the company's list of wines has something similar. Using the similarity matrix, we will look at the third row to find any wines having high correlation value (i.e., column value closer to 1). **The top 4 matches** of wine 3 in terms of chemical properties are wines 52, 51, 85, and 15. The chemical properties of the matching wines are shown below:

```
sim.items <- cor.matrix[3,]
# find the closest match by sorting row 3 in decreasing order
sim.items.sorted <- sort(sim.items, decreasing=TRUE)
# take out top 5 matches
sim.items.name <- names(sim.items.sorted[2:5])
rbind(wine.data[as.double(sim.items.name),])
```

```
##    V1    V2   V3   V4   V5  V6   V7   V8   V9  V10  V11  V12  V13  V14
## 1:  1 13.83 1.65 2.60 17.2  94 2.45 2.99 0.22 2.29 5.60 1.24 3.37 1265
## 2:  1 13.05 1.73 2.04 12.4  92 2.72 3.27 0.17 2.91 7.20 1.12 2.91 1150
## 3:  2 11.84 0.89 2.58 18.0  94 2.20 2.21 0.22 2.35 3.05 0.79 3.08  520
## 4:  1 14.38 1.87 2.38 12.0 102 3.30 3.64 0.29 2.96 7.50 1.20 3.00 1547
```

2 Case study: News aggregator

2.1 Problem statement

We have 1,000 news articles from different publishers in different categories: technical, entertainment, etc. The problem is building a recommendation system for anonymous customers. The customers either are first time visitors or we have not recorded their interaction with our products.

```
#
library(dplyr)
library(ggplot2)
library(sentimentr); library(sets); library(slam)
library(tidytext); library(tidyverse); library(tm)

data.population <- read_tsv('data\\NewsAggregatorDataset\\newsCorpora.csv',
                   col_names=c('ID','TITLE','URL','PUBLISHER','CATEGORY',
                               'STORY','HOSTNAME','TIMESTAMP'),
                   col_types=cols(
                     ID       =col_integer(),
                     TITLE    =col_character(),
                     URL      =col_character(),
                     PUBLISHER=col_character(),
                     CATEGORY =col_character(),
                     STORY    =col_character(),
                     HOSTNAME =col_character(),
                      TIMESTAMP=col_double()))

num.articles = 10 # number of related articles to recommend to the user
# sample without replacement 40,000 random data in the whole data set
index.sampled <- sample(1:nrow(data.population), 40000, replace=FALSE)
data.sample <- data.population[index.sampled,]
```

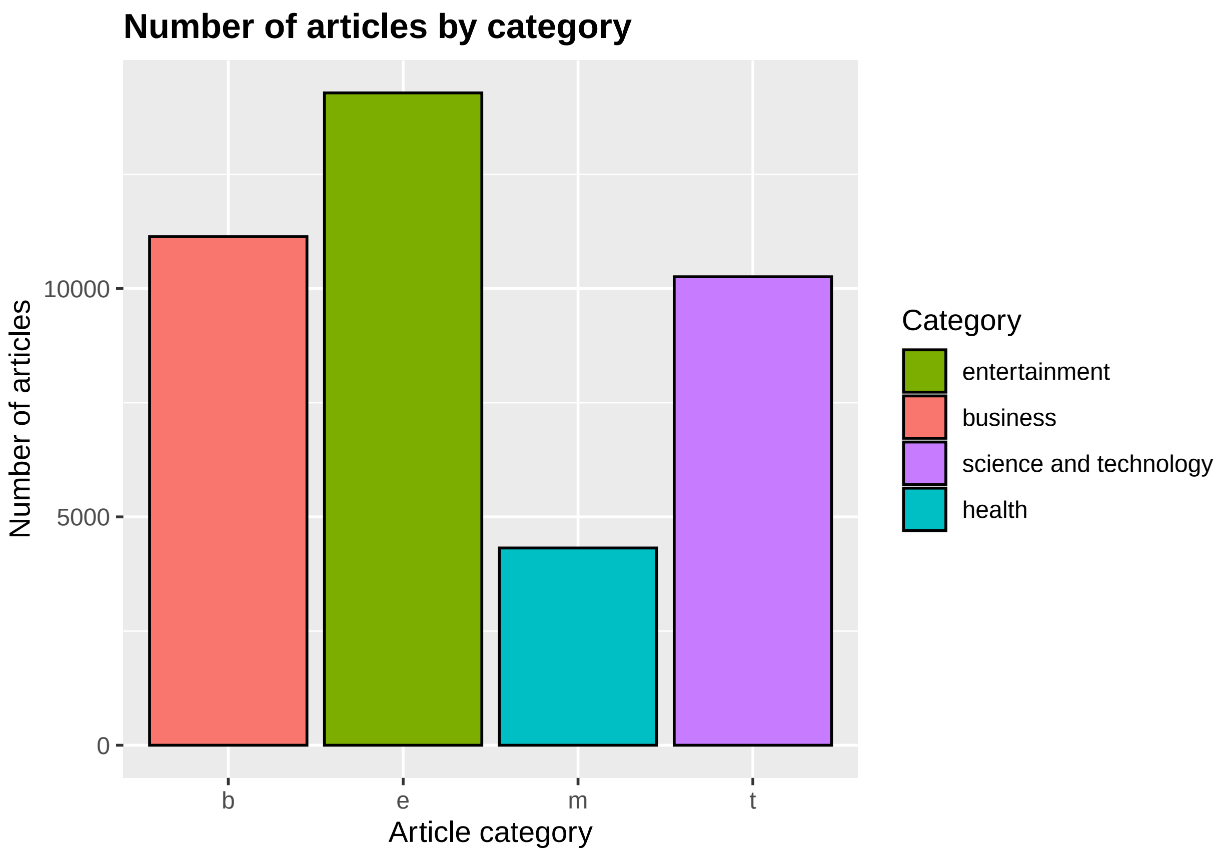
Every article has the following columns:

- `ID`: A unique identifier.
- `TITLE`: The title of the article (free text).
- `URL`: The article's URL.
- `PUBLISHER`: Publisher of the article.
- `CATEGORY`: Categorization under which the articles are grouped (`b` = business, `t` = science and technology, `e` = entertainment, `m` = health).
- `STORY`: An ID for the group of stories the article belongs to.
- `HOSTNAME`: Hostname of the URL.
- `TIMESTAMP`: Approximate time the news was published, as the number of milliseconds since the epoch 00:00:00 GMT, January 1, 1970.

2.2 Exploratory Data Analysis

Firstly, the articles are grouped by category (as specified by `CATEGORY`) and arranged in a descending order. From observation, entertainment has the highest number of published articles.

```
ggplot(data.sample) +
  geom_bar(mapping=aes(x=CATEGORY, fill=factor(CATEGORY)),
           colour ='black',
           stat   ='count') +
  ggtitle("Number of articles by category") +
  xlab("Article category") +
  ylab("Number of articles") +
  scale_fill_discrete(name  ='Category',
                      breaks=c('e','b','t','m'),
                      labels=c("entertainment","business",
                               "science and technology","health")) +
  theme(plot.title=element_text(lineheight=1, face="bold"))
```



Secondly, we order the publishers in descending order by the number of published articles. `Reuters` publishes the most articles, followed by `Businessweek`. Then, all articles of the top 100 publishers are obtained from `newsCorpora.csv` to design content-based recommendation engine. In this problem, a user chooses a random article in the top 100 publishers sampled data set `data.subset`.

```
publisher.top <- data.sample |> group_by(PUBLISHER) |>
        summarise(count=n()) |> arrange(desc(count), PUBLISHER)
publisher.top
```

```
## # A tibble: 5,473 x 2
##    PUBLISHER         count
##    <chr>             <int>
##   1 Reuters            383
##   2 Huffington Post    232
##   3 Businessweek       227
##   4 NASDAQ             221
....
##   9 Los Angeles Times  173
## 10 RTT News            167
## # i 5,463 more rows
```

```
# get top 100 publishers, join with article tables
data.subset <- inner_join(data.frame(publisher.top[1:100,]), data.sample)
data.subset$count = NULL
```

2.3 Content-based recommendation engine design

The problem statement from customer requirements is: **When a customer browses a particular article, what other articles should we suggest to him?** From the statement, some properties of the news articles must be analyzed to provide a similar content to the one the user is reading. Those properties are:

- The article's content.
- The article's publisher.
- The article's category.

2.4 Design procedure

Cosine distance compares words between two documents.

2.4.1

Jaccard distance measures the publishers and categories

1. Calculate **cosine distance**. The bag-of-words model is constructed from the sampled article data set. Then, a similarity index is created using the vector space model.
2. Search for relevant news articles in the similarity index. Given an article,
3. Implement a fuzzy ranking engine. The ranking engine finds the top 10 matches in a ranked order using the cosine distance, Jaccard distance, and Manhattan distance.
4.

2.5 Design process

2.5.1 Building similarity index

The code below creates a bag-of-words model using document term matrix representation. The model employs pre-processed documents of the sample `data.subset` to build a corpus.

```
# identify articles by their ID
corpus <- data.subset[,c('ID','TITLE')] |> rename(c(doc_id='ID',
                                                    text='TITLE')) |>
  DataframeSource() |> Corpus() |>
  # text transformation in 5 steps below
  tm_map(removePunctuation)          |>
  tm_map(removeNumbers)              |>
  tm_map(stripWhitespace)            |>
  tm_map(content_transformer(tolower)) |>
  tm_map(removeWords, stopwords("english"))

dtm <- DocumentTermMatrix(corpus, control=list(wordLength=c(3,20),
```

```r
                                              weighting="weightTfIdf"))
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 10665, terms: 16237)>>
## Non-/sparse entries: 78354/173089251
## Sparsity           : 100%
## Maximal term length: 200
## Sample             :
##        Terms
## Docs     apple dordgmauxoamqjwonqrwtkgam google new news report says update
....
##   421170     0    1
##   53848      0    0
##   73257     20   40
```

The articles list separated into two data frames: `title.df` and `others.df`. Data frame `title.df` stores article titles while `others.df` stores articles' ID, publisher, and category. Stemming algorithms are excluded in this problem. The documents are pre-processed in five steps. The text transformation process is summarized below:

1. `removePunctuation`: remove punctuation
2. `removeNumbers`: remove numbers
3. `stripWhitespace`: remove unnecessary white spaces
4. `content_transformer`: change words to upper or lower case, depending on the inputs. In this problem, the text is converted to lower case with `tolower` argument. In this way, algorithms can treat "Dog" and "dog" the same.
5. `stopwords`: remove common words in a language. Some examples of stop words in English content are *the*, *is*, *at*, *which*, *and*, *on*, etc.

The code below performs two tasks: calculate cosine similarity, stored in `sim.score` and randomly choose an article from the sample `data.subset`. The elements in `sim.score` matrix correspond to the cosine similarity between two documents.

```r
# calculate cosine similarity
sim.score <- tcrossprod_simple_triplet_matrix(dtm) /
  (sqrt(row_sums(dtm^2) %*% t(row_sums(dtm^2))))

# randomly choose an article from top 100 publisher sampled data set
user.read <- sim.score |> colnames() |> data.frame() |> sample_n(1)
user.article <- data.subset[data.subset$ID==user.read[1,1],]
t(user.article)
```

```
##           9943
## PUBLISHER "Us Magazine"
## ID        "333517"
## TITLE     "Zendaya Drops Out of Aaliyah Biopic: \"She Wants to Do It Right\""
## URL       "http://www.usmagazine.com/entertainment/news/zendaya-drops-out-aaliyah-l
## CATEGORY  "e"
## STORY     "dCUqPPKC7yO3uoMPA9l-Zt0lKtWqM"
## HOSTNAME  "www.usmagazine.com"
## TIMESTAMP "1.404099e+12"
```

The output of the code represents the article a user choose. In this problem, the user decided to read **Zendaya Drops Out of Aaliyah Biopic: "She Wants to Do It Right"** by **Us Magazine**. The recommendation engine is responsible for selecting top 10 similar articles to the article.

The calculating method for similarity index depends on the problem and objectives of the problem. The reasons cosine similarity is preferred compared to Pearson coefficient are:

- Cosine similarity is invariant to changes in magnitude of values since it only consider frequency of the words. This is not the case for Pearson coefficient.
- Cosine similarity neglects word order while Pearson coefficient is not. Thus, cosine similarity is suitable for *high-dimensional data* like words in a corpus.
- Cosine similarity is more robust to outliers or noise than Pearson's coefficient.

2.5.2 Ranking search results

For this step, the code below extract the IDs of top related articles to **Zendaya Drops Out of Aaliyah Biopic: "She Wants to Do It Right"**, excluding the article itself.

```
match.docs  <- sim.score[user.read[1,1],]
match.df    <- data.frame(ID              =names(match.docs),
                          cosine          =match.docs,
                          stringsAsFactors=FALSE)
match.df$ID <- as.integer(match.df$ID)
# organize in descending order, take the top articles, excluding the original
match.refined <- match.df |> arrange(desc(cosine)) |> head(num.articles+1)
```

Since the ID of article Zendaya Drops Out of Aaliyah Biopic: "She Wants to Do It Right" is 333517, row 333517 of `sim.score` is selected. The row is organized in descending order. Then, only 10 articles with the highest similarity index is are chosen (excluding the original article).

The code below calculates the sentiment score of the top 10 articles. Then, the scores are updated in column `polarity` of data frame `match.refined`.

```
# inner join with data.subset to include all information about the articles
match.refined <- match.refined |> inner_join(data.subset)
```

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

```
# calculate sentiment score of each sentence in top 10 articles
# using their titles
sentiment.score <- sentiment(match.refined$TITLE)       |>
                   group_by(element_id)                  |>
```

```
                         summarise(sentiment=mean(sentiment))
match.refined$polarity <- sentiment.score$sentiment
as_tibble(match.refined)
```

```
## # A tibble: 11 x 10
##         ID cosine PUBLISHER          TITLE URL   CATEGORY STORY HOSTNAME TIMESTAMP
##      <int>  <dbl> <chr>              <chr> <chr> <chr>    <chr> <chr>        <dbl>
##  1 333517  1      Us Magazine        "Zen~ http~ e        dCUq~ www.usm~   1.40e12
##  2 333508  0.730 Hollywood Life     "Zen~ http~ e        dCUq~ hollywo~   1.40e12
##  3 337882  0.667 Entertainmentwise "Zen~ http~ e        dm8D~ www.ent~   1.40e12
##  4 333502  0.544 Contactmusic.com   "Lif~ http~ e        dCUq~ www.con~   1.40e12
##  5 333488  0.5    CTV News           "Tee~ http~ e        dCUq~ www.ctv~   1.40e12
##  6 333486  0.463 Examiner.com       "Con~ http~ e        dCUq~ www.exa~   1.40e12
##  7 333497  0.463 The Stir           "Zen~ http~ e        dCUq~ thestir~   1.40e12
##  8 333482  0.433 The Inquisitr      "Zen~ http~ e        dCUq~ www.inq~   1.40e12
##  9 300662  0.365 Daily Digest       "Zen~ http~ e        d9kn~ dailydi~   1.40e12
## 10 337852  0.333 BBC News           "Dis~ http~ e        dm8D~ www.bbc~   1.40e12
## 11  39972  0.289 Belfast Telegraph "Joh~ http~ e        dW_h~ www.bel~   1.40e12
## # i 1 more variable: polarity <dbl>
```

The function `sentiment` calculates the sentiment score of the top 10 related articles based on their titles. Since the calculation applies to each sentence, if an article title has more than one sentence, it may have two sentiment scores and above. Thus, the mean value is introduced to evaluate the overall score of each title.

The code below calculates the average value of publisher and category Jaccards index. We only concern if the publisher and category of related articles match Us Magazine and e, respectively. Thus, the final Jaccard index are the average value of two binary outputs.

```
# Jaccard index of publisher
match.refined$is.publisher <- as.numeric(match.refined$PUBLISHER==
                                               user.article$PUBLISHER)
# Jaccard index of category
match.refined$is.category  <- as.numeric(match.refined$CATEGORY==
                                               user.article$CATEGORY)
# average Jaccard value
match.refined$jaccard <- (match.refined$is.publisher + match.refined$is.category)/2
```

The code below calculates the normalized absolute difference between the target article and related articles:

```
# absolute difference (Manhattan distance)
match.refined$polaritydiff <- abs(match.refined[1,]$polarity -
                                    match.refined$polarity)
# min-max normalization
f_rescaling <- function(x){(x-min(x)) / (max(x)-min(x))}
match.refined$polaritydiff <- f_rescaling(match.refined$polaritydiff)

# clean up
```

```
match.refined$is.publisher = NULL
match.refined$is.category  = NULL
match.refined$polarity     = NULL
match.refined$sentiment    = NULL
match.refined$STORY        = NULL
match.refined$HOSTNAME     = NULL
match.refined$URL          = NULL
tibble(match.refined)
```

```
## # A tibble: 11 x 8
##        ID cosine PUBLISHER         TITLE CATEGORY TIMESTAMP jaccard polaritydiff
##     <int>  <dbl> <chr>             <chr> <chr>        <dbl>   <dbl>        <dbl>
##  1 333517  1     Us Magazine       "Zen~ e          1.40e12     1          0
##  2 333508  0.730 Hollywood Life    "Zen~ e          1.40e12     0.5        0.634
##  3 337882  0.667 Entertainmentwise "Zen~ e          1.40e12     0.5        0.634
##  4 333502  0.544 Contactmusic.com  "Lif~ e          1.40e12     0.5        0.444
##  5 333488  0.5   CTV News          "Tee~ e          1.40e12     0.5        0.634
##  6 333486  0.463 Examiner.com      "Con~ e          1.40e12     0.5        0.721
##  7 333497  0.463 The Stir          "Zen~ e          1.40e12     0.5        0.863
##  8 333482  0.433 The Inquisitr     "Zen~ e          1.40e12     0.5        1
##  9 300662  0.365 Daily Digest      "Zen~ e          1.40e12     0.5        0.634
## 10 337852  0.333 BBC News          "Dis~ e          1.40e12     0.5        0.634
## 11  39972  0.289 Belfast Telegraph "Joh~ e          1.40e12     0.5        0.634
```

After calculation, all unnecessary columns are removed to rank the related articles.

2.5.3 Ranking Search Results using Fuzzy Logic

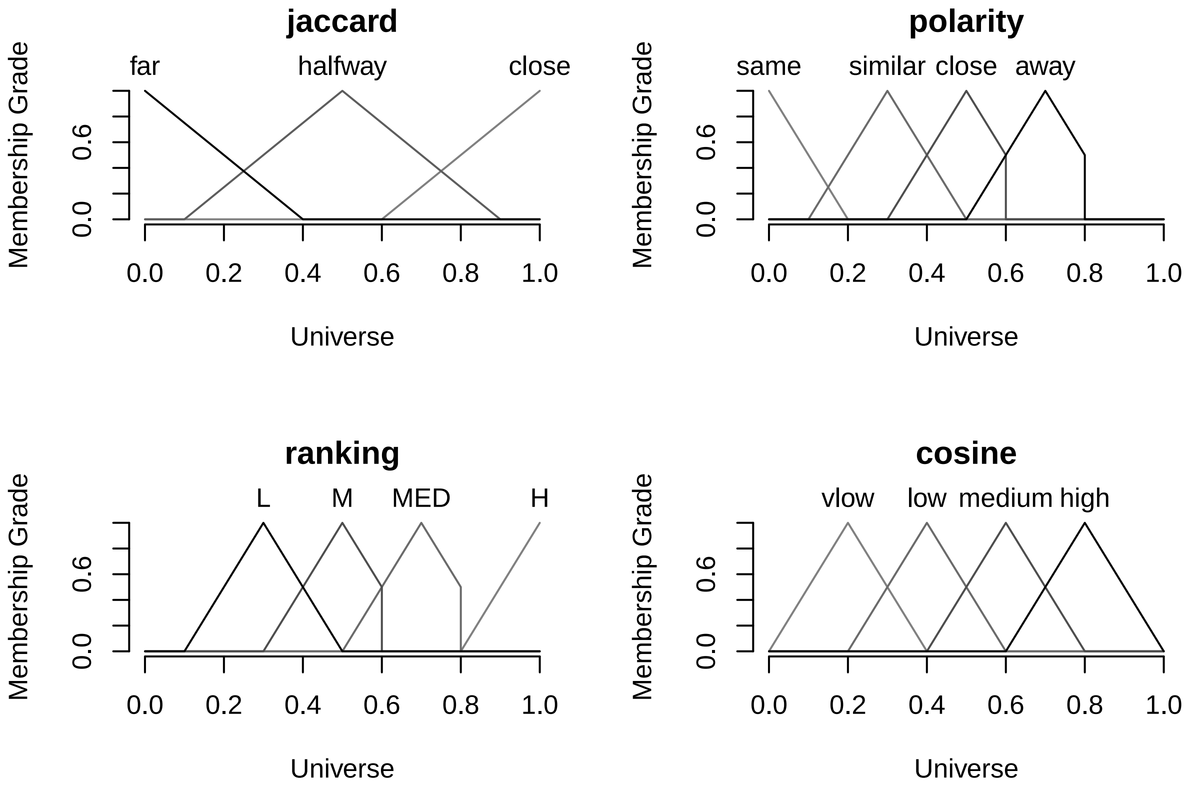
```
  #### Fuzzy Logic ####
sets_options("universe", seq(from=0, to=1, by=0.1))
variables <- set(
  cosine=
  fuzzy_partition(varnames=c(vlow=0.2, low=0.4, medium=0.6, high=0.8),
                      FUN=fuzzy_cone, radius=0.2),
  jaccard=
  fuzzy_partition(varnames=c(close=1.0, halfway=0.5, far=0.0),
                      FUN=fuzzy_cone , radius=0.4),
  polarity=
  fuzzy_partition(varnames=c(same=0.0, similar=0.3, close=0.5, away=0.7),
                      FUN = fuzzy_cone , radius = 0.2),
  ranking =
  fuzzy_partition(varnames=c(H=1.0, MED=0.7, M=0.5, L=0.3),
                      FUN = fuzzy_cone , radius = 0.2)
  )
```

```r
rules <- set(
  #### Fuzzy Rule ####
  # 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 Rules
  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)
  )

  #### ako ####
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% M
## cosine %is% medium && jaccard %is% halfway && polarity %is% same => ranking %is% ME
## cosine %is% medium && jaccard %is% halfway && polarity %is% similar => ranking %is%
## 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
```

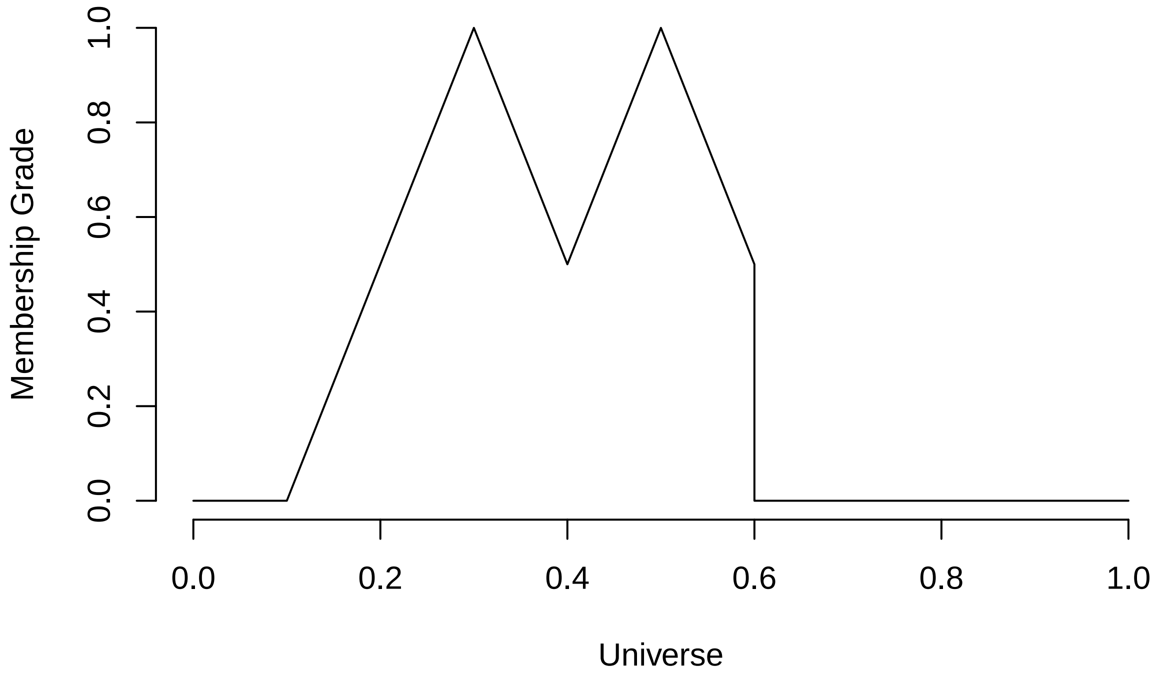
```r
plot(ranking.system)
```



```
fi <- fuzzy_inference(ranking.system, list(cosine=0.5, jaccard=0, polarity=0.0))
gset_defuzzify(fi, "centroid")
```

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

```
plot(fi)
```



```
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),]
as_tibble(match.refined)
```

```
## # A tibble: 11 x 9
##        ID cosine PUBLISHER TITLE CATEGORY TIMESTAMP jaccard polaritydiff ranking
##     <int>  <dbl> <chr>     <chr> <chr>        <dbl>   <dbl>        <dbl>   <dbl>
##   1 333517 1     Us Magaz~ "Zen~ e          1.40e12     1            0       0.4
##   2 333488 0.5   CTV News  "Tee~ e          1.40e12     0.5          0.634   0.383
##   3 333508 0.730 Hollywoo~ "Zen~ e          1.40e12     0.5          0.634   0.3
##   4 337882 0.667 Entertai~ "Zen~ e          1.40e12     0.5          0.634   0.3
##   5 333502 0.544 Contactm~ "Lif~ e          1.40e12     0.5          0.444   0.3
##  6 333486  0.463 Examiner~ "Con~ e          1.40e12     0.5        0.721   0.3
##  7 333497  0.463 The Stir  "Zen~ e          1.40e12     0.5        0.863   0.3
##  8 333482  0.433 The Inqu~ "Zen~ e          1.40e12     0.5        1       0.3
##  9 300662  0.365 Daily Di~ "Zen~ e          1.40e12     0.5        0.634   0.3
## 10 337852  0.333 BBC News  "Dis~ e          1.40e12     0.5        0.634   0.3
## 11  39972  0.289 Belfast ~ "Joh~ e          1.40e12     0.5        0.634   0.3
```

3 Appendix 1: Self-noted terminologies

There are two different ways to collect samples: **Sampling with replacement** and **sampling without replacement**.

- **Sampling with replacement** is the method where the items in the samples are *independent* because the outcome of one random draw is not affected by the previous draw. Sampling with replacement is useful in many different scenarios in statistics and machine learning like *bootstrapping*, *bagging*, *boosting*, *random forests*, etc. The sampling method allows generating different models with the same data set. This is less time-consuming and expensive than acquiring new data every time a new model is built.
- **Sampling without replacement** is the method where the items in the samples are *dependent* because the outcome of one random draw is affected by the previous draw. It is typically useful when we want to select a random sample from a population.

3.1 Feature engineering

Polarity of the document. A subjective content about a topic tends to have a *positive*, *negative*, or *neutral* perspective. Polarity identification algorithms quantify the perspective using text mining. Manhattan distance is a simple algorithm to measure the polarity value.

In data analysis, **cosine similarity** is a *measure of similarity* between two non-zero vectors defined in an inner product space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belongs to the interval $[-1, 1]$. The term cosine distance is commonly used for the complement of cosine similarity in positive space, that is

$$\text{cosine distance} = D_C(A, B) := 1 - S_C(A, B) = 1 - \frac{\sum_{i=1} n(A_i B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The **Jaccard index**, also known as the **Jaccard similarity coefficient**, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. The

Jaccard distance, which measures *dis*similarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union:

$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

3.1.1 Bag-of

In a **bag-of-words model**, every document is represented as a collection of words. In the model, the order of words is irrelevant. A good way to organize these bag-of-words models is using a **document term matrix**.

A **document term matrix** is a matrix representation of words in a collection of writings. For instance, we have 100 documents, which is called a *corpus*. In order to build the matrix, we first list out all unique words in 100 documents, which is called a *vocabulary*. If there are 5,000 unique words in 100 documents, the matrix's dimension is 100 x 5,000, where the rows are the documents and the columns are the words in the vocabulary.

Another example, considering a corpus of *sentences*:

1. **Sentence 1**: Dog chased cat.
2. **Sentence 2**: Dog hates cat.

The document term matrix represents the bag-of-words model of the 2 sentences, which is shown below:

Document/word	hates	dog	chased	cat
Sentence 1	0	1	1	1
Sentence 2	1	1	0	1

Term frequency is the number of times a word appears in a document. For example, the term frequency of the word "dog" in sentence 1 above is 1, with a normalization of 1/3. Another example is an article that has 500 words, 25 of which mention the word "dog". The term frequency is 25, or 0.05 if it is normalized.

Document frequency is the number of documents in the corpus that contain a word. In the example above, the document frequency of the word "dog" in the corpus is 2.

Inverse document frequency (IDF) measures how important a word is to a document. The higher the IDF, the more rare and informative the word is.

$$IDF(w) = \frac{N}{df(w)}$$

TFIDF is the product of term frequency (TF) and inverse document frequency (IDF). It is a commonly used weighting metric in text mining. The formula for TFIDF is:

$$TFIDF(w, d) = tf(w, d)IDF(w)$$

where `tf(w,d)` is the term frequency; `IDF(w)` is the inverse document frequency.

4 Bibliography

Sampling With Replacement vs. Without Replacement

How to Normalize Data in R - Statology