DATA SOCIETY®

Week 6 Day 2 - Text Data Analysis

"One should look for what is and not what he thinks should be."
-Albert Einstein.

"Bag-of-words" analysis: Review

Text preparation and cleaning is one of the most important steps in text mining

1. Convert all characters to lower case
2. Remove stop words
3. Remove punctuation
4. Remove numbers & all other symbols that are not letters of the alphabet
5. Stem words
6. Remove extra whitespace
7. Create a Term-Document matrix

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	
Introduce the concept of cosine similarity and distance	
Compute term similarity matrix for wiki corpus and transform it into tidy data format	
Create wiki corpus term similarity heatmap	
Compare terms by trimming cosine similarity data and building a network graph	
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Text mining workflow

Text analysis workflow

Summarize

Reduce

Weight

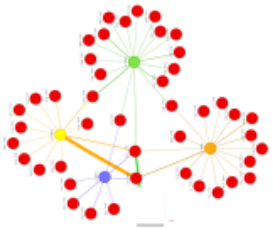
Index	Word	Weighted freq
A	Apple	0.75
B	Book	0.1
C	Cat	0.025

Compute similarity

Word1	Word2	Score
Book	Apple	0.5
Book	Book	1
Book	Cat	0.3

Visualize similar documents/terms

Perform other ML methods on text data



Directory settings

- First, let's make sure to set our directories correctly.

```
# Set `main_dir` to the location of your `hhs-r` folder (for Mac/Linux).
main_dir = "~/Desktop/hhs-r-2020"
# Set `main_dir` to the location of your `hhs-r` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/hhs-r-2020"
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Load packages

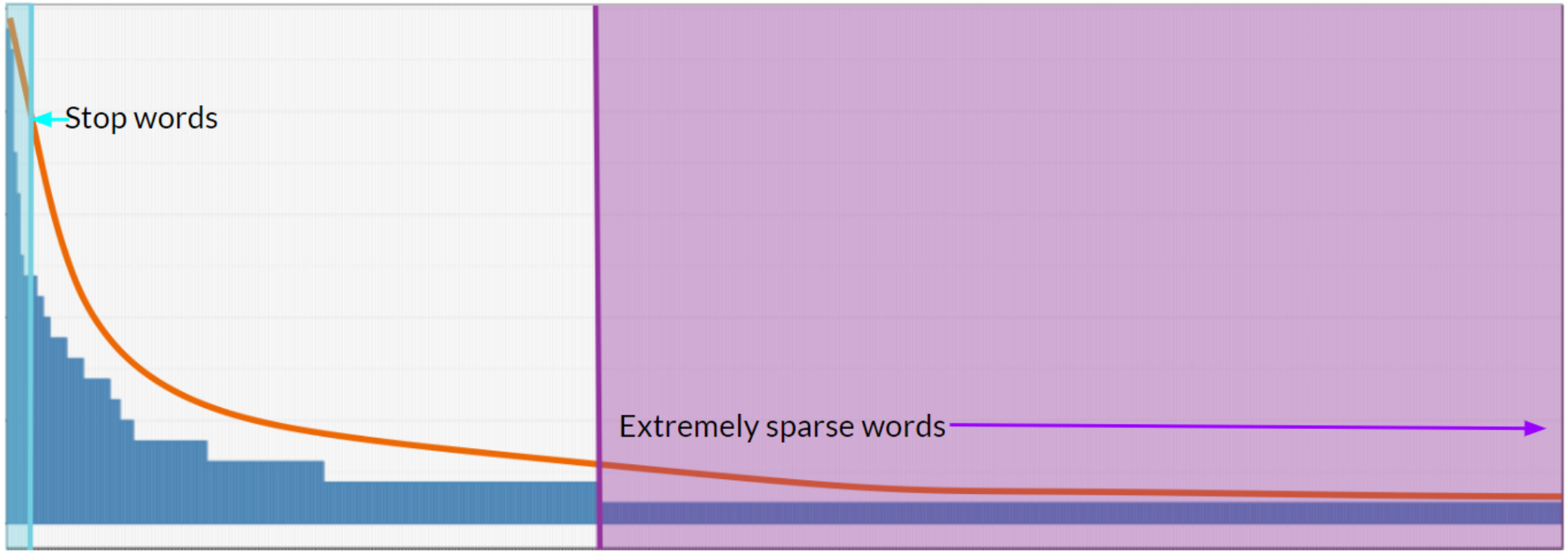
```
# Load `tm` library.
library(tm)

# Load tidyverse.
library(tidyverse)
library(broom)

# Install `visNetwork` package.
# install.packages("visNetwork")
library(visNetwork)
```

Load nwiki and arXiv corpus

```
# Set working directory to `data_dir`.
setwd(data_dir)
# Load wiki corpus.
load("wiki_corpus_clean.RData")
```

Word distribution in a language and corpus

- Distribution of words in a corpus and in a language is **highly skewed to the right**
- Words that have extremely high frequencies are considered **stop words**
- Those that have extremely low frequencies are considered **extremely sparse words**

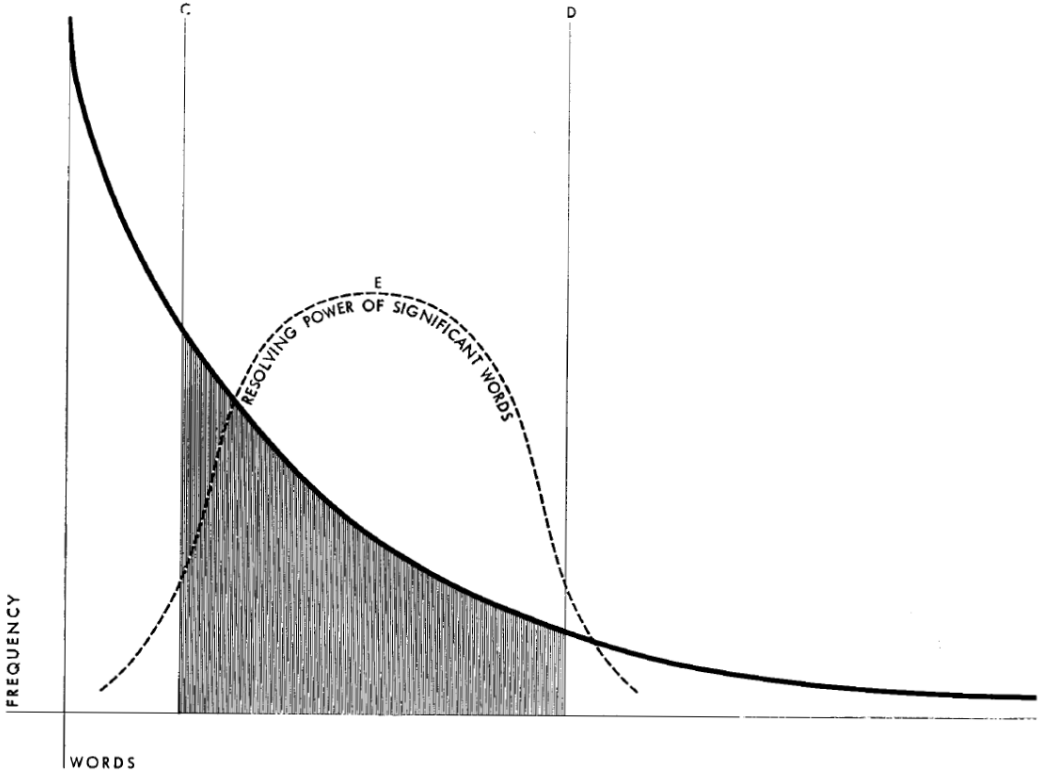


Power of significant words

- By removing *BOTH* ends of the distribution we **reduce the dimensionality** of our data and **avoid "overfitting"** of our text model

- In fact, it has been proven by *H.P. Luhn* (a researcher for IBM) *in 1958* that the **power of significant words is approximately normally distributed** and the top of the bell-shaped curve **coincides with the mid portion of the word frequency distribution**

Figure 1 **Word-frequency diagram.**
Abscissa represents individual words arranged in order of frequency.



Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts", *Presented at IRE National Convention, New York, March 24, 1958. Published in IBM JOURNAL APRIL 1958*

Why bother?

In order to have **quality** calculations that measure **term or document similarity**:

- **Avoid** having **too many dimensions**
- Nearly identical objects lead to **strange side effects in numerical computation**
- **Speed up** computations

DTM vs TDM

When we talk about dimensionality in text data, we are working with 2 types of data:

1. Data where **documents** are the observations and **terms** are the variables; use **DTM** when comparing **documents**
2. Data where **terms** are the observations and **documents** are the variables; use **TDM** when comparing **terms**

In which case do you think term trimming will reduce the number of dimensions?

Terms are in columns

Documents are in rows

	abstract	academ	acquaint	action	activ	actor	addit	affect	alert	algebra	analysi	analyt
Doc 1	0	0	0	0	0	0	0	0	0	1	0	0
Doc 2	1	0	0	0	0	0	0	0	0	7	1	1
Doc 3	0	0	0	0	0	0	0	0	0	0	0	0
Doc 4	0	0	0	0	0	0	1	0	0	0	5	3
Doc 5	0	0	1	0	0	1	0	0	0	0	3	0
Doc 6	0	1	0	0	1	0	0	1	0	0	3	2
Doc 7	0	0	0	1	0	0	0	0	2	0	1	5

Documents are in columns

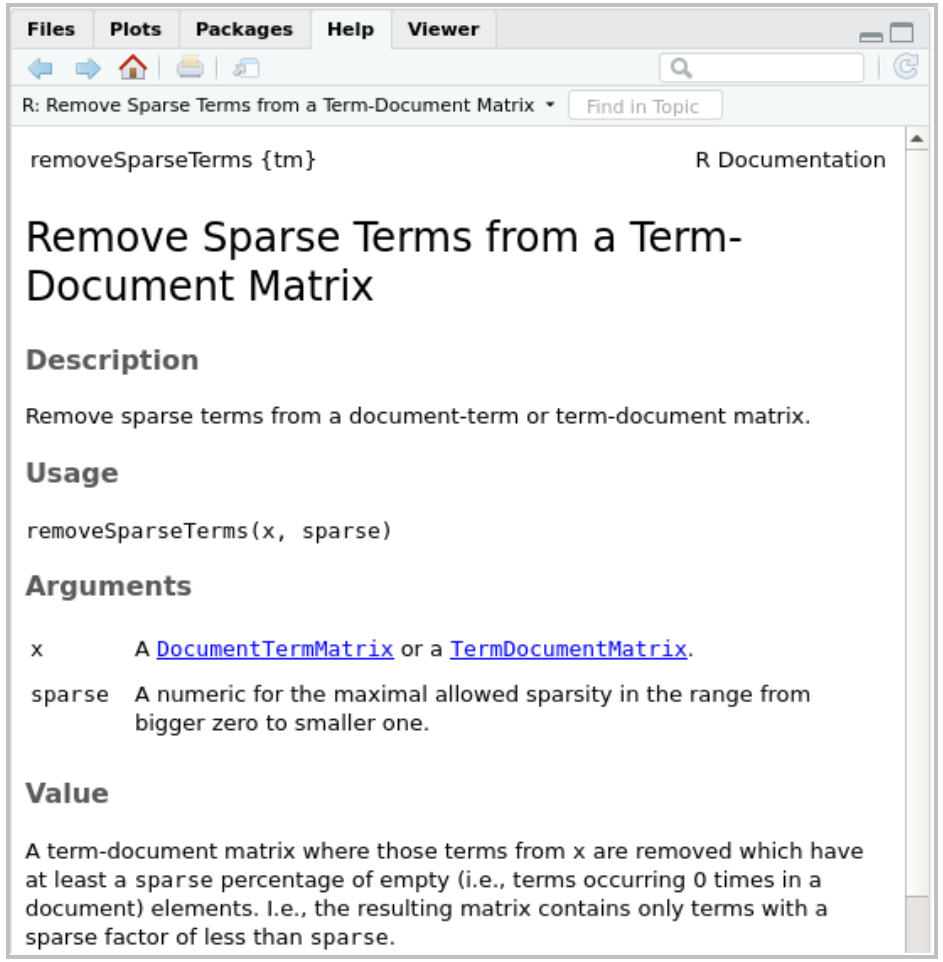
Terms are in rows

	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7
abstract	0	1	0	0	0	0	0
academ	0	0	0	0	0	1	0
acquaint	0	0	0	0	1	0	0
action	0	0	0	0	0	0	1
activ	0	0	0	0	0	1	0
actor	0	0	0	0	1	0	0
addit	0	0	0	1	0	0	0
affect	0	0	0	0	0	1	0
alert	0	0	0	0	0	0	2
algebra	1	7	0	0	0	0	0
analysi	0	1	0	5	3	3	1
analyt	0	1	0	3	0	2	5

Dealing with sparsity

```
?removeSparseTerms

removeSparseTerms(x,        #<- TDM or DTM object
                  sparse)   #<- max sparsity
```

- **Threshold** for determining **sparsity** is **different** for every corpus and usually it takes a few iterations to determine the optimal
- Max sparsity here refers to the threshold of **relative document frequency** for a term, above which the term will be **removed.**
- Relative document frequency means a proportion, so **sparsity is smaller** as it approaches **1.0.**



Files | Plots | Packages | Help | Viewer

R: Remove Sparse Terms from a Term-Document Matrix Find in Topic

removeSparseTerms {tm} R Documentation

Remove Sparse Terms from a Term-Document Matrix

Description

Remove sparse terms from a document-term or term-document matrix.

Usage

```
removeSparseTerms(x, sparse)
```

Arguments

x	A DocumentTermMatrix or a TermDocumentMatrix.
sparse	A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one.

Value

A term-document matrix where those terms from x are removed which have at least a sparse percentage of empty (i.e., terms occurring 0 times in a document) elements. I.e., the resulting matrix contains only terms with a sparse factor of less than sparse.

Removing sparse terms in wiki corpus

```
# Construct a term document matrix.
wiki_TDM = TermDocumentMatrix(wiki_corpus_clean)
wiki_TDM
```

```
<<TermDocumentMatrix (terms: 466, documents:
7)>>
Non-/sparse entries: 629/2633
Sparsity           : 81%
Maximal term length: 17
Weighting          : term frequency (tf)
```

Observe that current sparsity is 81%

Let's set the sparsity threshold to a maximum of `0.75` and remove the sparse terms

```
# Remove sparse terms from a TDM.
wiki_TDM = removeSparseTerms(wiki_TDM,
                             sparse = 0.75)
wiki_TDM
```

```
<<TermDocumentMatrix (terms: 102, documents:
7)>>
Non-/sparse entries: 265/449
Sparsity           : 63%
Maximal term length: 10
Weighting          : term frequency (tf)
```

Module completion checklist

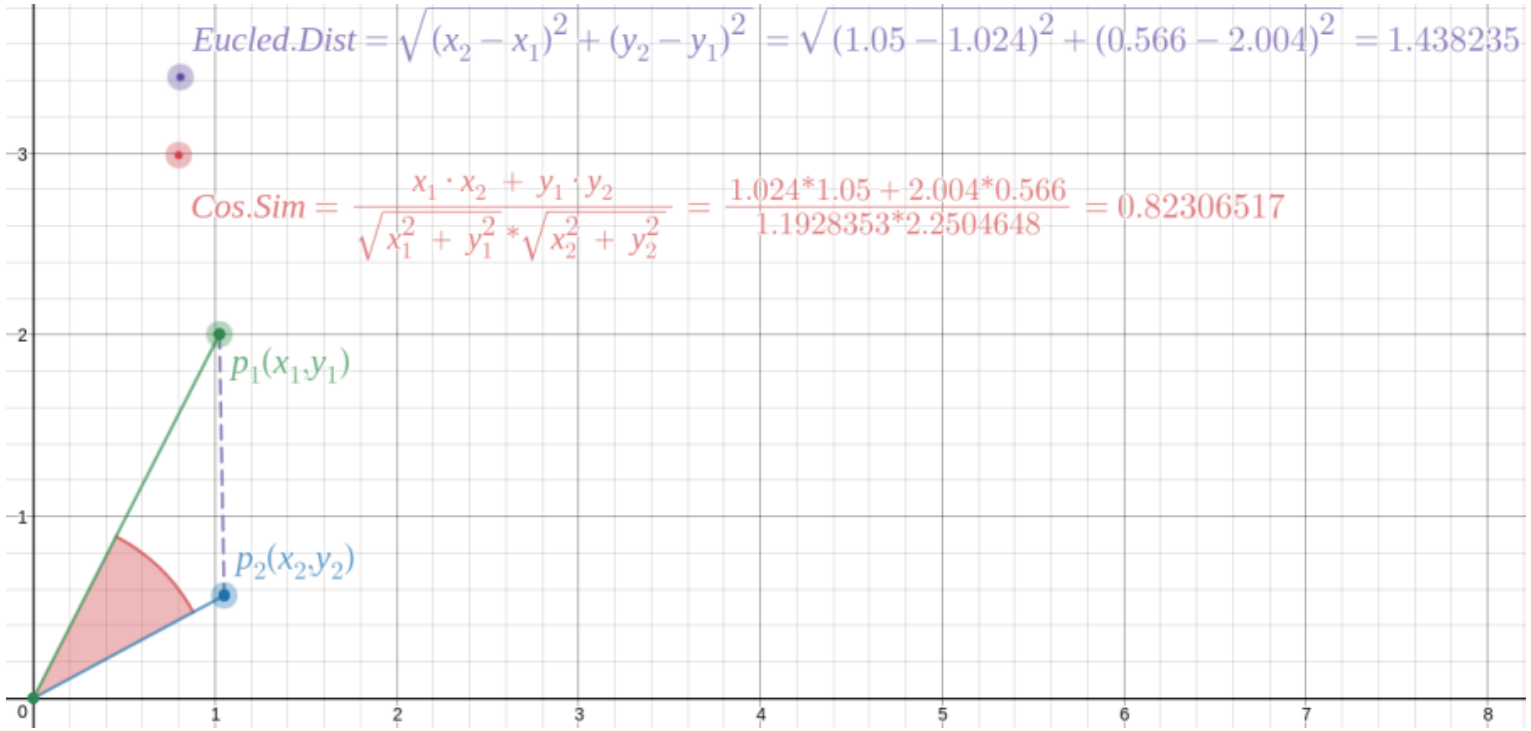
Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	
Compute term similarity matrix for wiki corpus and transform it into tidy data format	
Create wiki corpus term similarity heatmap	
Compare terms by trimming cosine similarity data and building a network graph	
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Comparing terms and documents in text

- One of the most common analyses in text mining is **finding similar terms and documents**
- This allows us to sort large amounts of documents or terms and instead of using exact search, **use contextual comparison** instead
- Just like with any other two data points, to find **two terms or documents** that are **similar** in their meaning we need to **measure the distance** between them

Measuring similarity and distance in text

- When working with text data, a different similarity metric is used, it's called **cosine similarity**
- **Cosine similarity** is $cos(a)$, where a is an angle between the two *vectors*
- **Cosine distance** = 1 - **cosine similarity**
- Note that cosine similarity & cosine distance will always be between `[0,1]`



Interpreting similarity and distance in text

Measure	Value	Meaning
Cosine Similarity	0	stands for two observations being orthogonal (i.e. being at 90-degree angle, completely dissimilar)
Cosine Similarity	1	stands for two observations having an angle of 0 and overlapping vectors
Cosine Distance	0	will stand for the distance between identical points
Cosine Distance	1	will stand for the most dissimilar of observations

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	
Create wiki corpus term similarity heatmap	
Compare terms by trimming cosine similarity data and building a network graph	
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Cosine similarity function

- We have prepared a function for you that computes **cosine similarity** for a `TDM` or a `DTM` matrix
- We are not going to be covering computational specifics of this function in detail, but you are welcome to explore the `tcrossprod` function for matrix multiplication

```
# Cosine similarity function.
CosineSim = function(x){
  sums_of_squares = rowSums(x ^ 2)
  sim = tcrossprod(x) / sqrt(tcrossprod(sums_of_squares))

  return(sim)
}
```

- **Input:** `x` is an `n x m` matrix for which you want to compute the cosine similarity
- **Output:** `sim` is an `n x n` *square and symmetric* matrix of similarity scores, that has `1`s across the diagonal

Measuring similarity of terms

- To measure the similarity of `terms` in our corpus, we need to pass `TDM` to a `CosineSim` function
- Since our `wiki_TDM` has `102 terms` and `7 documents`, the similarity matrix should be `102 x 102`

```
# Convert `TDM` object to a matrix.
wiki_TDM_matrix = as.matrix(wiki_TDM)

# Compute cosine similarity for TDM matrix.
wiki_term_sim = CosineSim(wiki_TDM_matrix)

# Take a look at the structure of the matrix.
str(wiki_term_sim)
```

```
 num [1:102, 1:102] 1 0.345 0.148 0.14 0.653 ...
 - attr(*, "dimnames")=List of 2
  ..$ Terms: chr [1:102] "algebra" "also" "analysi" "analyt" ...
  ..$ Terms: chr [1:102] "algebra" "also" "analysi" "analyt" ...
```

Term similarity matrix

```
# View the term
# similarity matrix.
View(wiki_term_sim)
```

- We can see **terms** in both **rows** and **columns**
- **1s on the diagonal**
- **Symmetric** around the diagonal

wiki_term_sim

Filter

	algebra	also	analysi	analyt	appli	area	associ	base	biolog	busi	can	combin	common	c
algebra	1.00000000	0.3446562	0.14757296	0.14000000	0.65319726	0.00000000	0.1000000	0.00000000	0.00000000	0.00000000	0.06030227	0.70000000	0.44271887	
also	0.34465617	1.0000000	0.36329951	0.32003788	0.30151134	0.73854895	0.1230915	0.54494926	0.24618298	0.20483662	0.66804266	0.36927447	0.31139958	
analysi	0.14757296	0.3632995	1.00000000	0.59029183	0.34426519	0.31622777	0.5270463	0.13333333	0.63245553	0.23388214	0.09534626	0.63245553	0.46666667	
analyt	0.14000000	0.3200379	0.59029183	1.00000000	0.24494897	0.20000000	0.3000000	0.75894664	0.20000000	0.88752031	0.54272042	0.40000000	0.06324555	
appli	0.65319726	0.3015113	0.34426519	0.24494897	1.00000000	0.40824829	0.4082483	0.00000000	0.40824829	0.11322770	0.24618298	0.40824829	0.25819889	
area	0.00000000	0.7385489	0.31622777	0.20000000	0.40824829	1.00000000	0.0000000	0.31622777	0.50000000	0.13867505	0.45226702	0.00000000	0.00000000	
associ	0.10000000	0.1230915	0.52704628	0.30000000	0.40824829	0.00000000	1.0000000	0.00000000	0.00000000	0.00000000	0.30151134	0.50000000	0.00000000	
base	0.00000000	0.5449493	0.13333333	0.75894664	0.00000000	0.31622777	0.0000000	1.00000000	0.00000000	0.87705802	0.85811633	0.00000000	0.00000000	
biolog	0.00000000	0.2461830	0.63245553	0.20000000	0.40824829	0.50000000	0.0000000	0.00000000	1.00000000	0.13867505	0.00000000	0.00000000	0.63245553	
busi	0.00000000	0.2048366	0.23388214	0.88752031	0.11322770	0.13867505	0.0000000	0.87705802	0.13867505	1.00000000	0.62718151	0.00000000	0.00000000	
can	0.06030227	0.6680427	0.09534626	0.54272042	0.24618298	0.45226702	0.3015113	0.85811633	0.00000000	0.62718151	1.00000000	0.00000000	0.00000000	
combin	0.70000000	0.3692745	0.63245553	0.40000000	0.40824829	0.00000000	0.5000000	0.00000000	0.00000000	0.00000000	0.00000000	1.00000000	0.31622777	
common	0.44271887	0.3113996	0.46666667	0.06324555	0.25819889	0.00000000	0.0000000	0.00000000	0.63245553	0.00000000	0.00000000	0.31622777	1.00000000	
comput	0.54912518	0.3379632	0.45479403	0.31378582	0.80064077	0.58834841	0.0000000	0.00000000	0.58834841	0.16317849	0.00000000	0.39223227	0.24806947	
condit	0.80000000	0.2461830	0.10540926	0.10000000	0.81649658	0.00000000	0.5000000	0.00000000	0.00000000	0.00000000	0.30151134	0.50000000	0.31622777	
continu	0.00000000	0.5685352	0.30429031	0.80829038	0.23570226	0.57735027	0.0000000	0.91287093	0.28867513	0.88070485	0.78334945	0.00000000	0.00000000	
data	0.00000000	0.2252553	0.61084091	0.45749571	0.56031553	0.68624357	0.1524986	0.09644856	0.68624357	0.29606845	0.06897007	0.15249857	0.00000000	
databas	0.00000000	0.2461830	0.84327404	0.50000000	0.40824829	0.50000000	0.5000000	0.00000000	0.50000000	0.13867505	0.00000000	0.50000000	0.00000000	

Showing 1 to 19 of 102 entries

Converting symmetric matrix to triangular

- Since the matrix is symmetric, we really only need either its lower or the upper triangle
- R base function `as.dist` converts a symmetric matrix into a `dist` class, which only retains one half of its entries excluding the diagonal entries

```
?as.dist
```

Filter

	algebra	also	analysi	analyt	appli	area	associ	base	biolog	busi	can	combin	common
algebra	1.00000000	0.3446562	0.14757296	0.14000000	0.65319726	0.00000000	0.1000000	0.00000000	0.00000000	0.00000000	0.06030227	0.70000000	0.44271887
also	0.34465617	1.0000000	0.36329951	0.32003788	0.30151134	0.73854895	0.1230915	0.54494926	0.24618298	0.20483662	0.66804266	0.36927447	0.31139958
analysi	0.14757296	0.3632995	1.00000000	0.59029183	0.34426519	0.31622777	0.5270463	0.13333333	0.63245553	0.23388214	0.09534626	0.63245553	0.46666667
analyt	0.14000000	0.3200379	0.59029183	1.00000000	0.24494897	0.20000000	0.3000000	0.75894664	0.20000000	0.88752031	0.54272042	0.40000000	0.06324555
appli	0.65319726	0.3015113	0.34426519	0.24494897	1.00000000	0.40824829	0.4082483	0.00000000	0.40824829	0.11322770	0.24618298	0.40824829	0.25819889
area	0.00000000	0.7385489	0.31622777	0.20000000	0.40824829	1.00000000	0.0000000	0.31622777	0.50000000	0.13867505	0.45226702	0.00000000	0.00000000
associ	0.10000000	0.1230915	0.52704628	0.30000000	0.40824829	0.00000000	1.0000000	0.00000000	0.00000000	0.00000000	0.30151134	0.50000000	0.00000000
base	0.00000000	0.5449493	0.13333333	0.75894664	0.00000000	0.31622777	0.0000000	1.00000000	0.00000000	0.87705802	0.85811633	0.00000000	0.00000000
biolog	0.00000000	0.2461830	0.63245553	0.20000000	0.40824829	0.50000000	0.0000000	0.00000000	1.00000000	0.13867505	0.00000000	0.00000000	0.63245553
busi	0.00000000	0.2048366	0.23388214	0.88752031	0.11322770	0.13867505	0.0000000	0.87705802	0.13867505	1.00000000	0.62718151	0.00000000	0.00000000
can	0.06030227	0.6680427	0.09534626	0.54272042	0.24618298	0.45226702	0.3015113	0.85811633	0.00000000	0.62718151	1.00000000	0.00000000	0.00000000
combin	0.70000000	0.3692745	0.63245553	0.40000000	0.40824829	0.00000000	0.5000000	0.00000000	0.00000000	0.00000000	0.00000000	1.00000000	0.31622777
common	0.44271887	0.3113996	0.46666667	0.06324555	0.25819889	0.00000000	0.0000000	0.00000000	0.63245553	0.00000000	0.00000000	0.31622777	1.00000000
comput	0.54912518	0.3379632	0.45479403	0.31378582	0.80064077	0.58834841	0.0000000	0.00000000	0.58834841	0.16317849	0.00000000	0.39223227	0.24806947

Files | Plots | Packages | Help | Viewer

R: Distance Matrix Computation Find in Topic

dist {stats} R Documentation

Distance Matrix Computation

Description

This function computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.

Usage

```
dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)

as.dist(m, diag = FALSE, upper = FALSE)
## Default S3 method:
as.dist(m, diag = FALSE, upper = FALSE)

## S3 method for class 'dist'
print(x, diag = NULL, upper = NULL,
      digits = getOption("digits"), justify = "none",
      right = TRUE, ...)

## S3 method for class 'dist'
as.matrix(x, ...)
```

Arguments

x	a numeric matrix, data frame or "dist" object.

Term similarity matrix: convert to dist

```
# Convert similarity matrix to a `dist` object to keep only half of its entries.
wiki_term_sim_half = as.dist(wiki_term_sim)
str(wiki_term_sim_half)
```

```
 'dist' num [1:5151] 0.345 0.148 0.14 0.653 0 ...
 - attr(*, "Labels")= chr [1:102] "algebra" "also" "analysi" "analyt" ...
 - attr(*, "Size")= int 102
 - attr(*, "call")= language as.dist.default(m = wiki_term_sim)
 - attr(*, "Diag")= logi FALSE
 - attr(*, "Upper")= logi FALSE
```

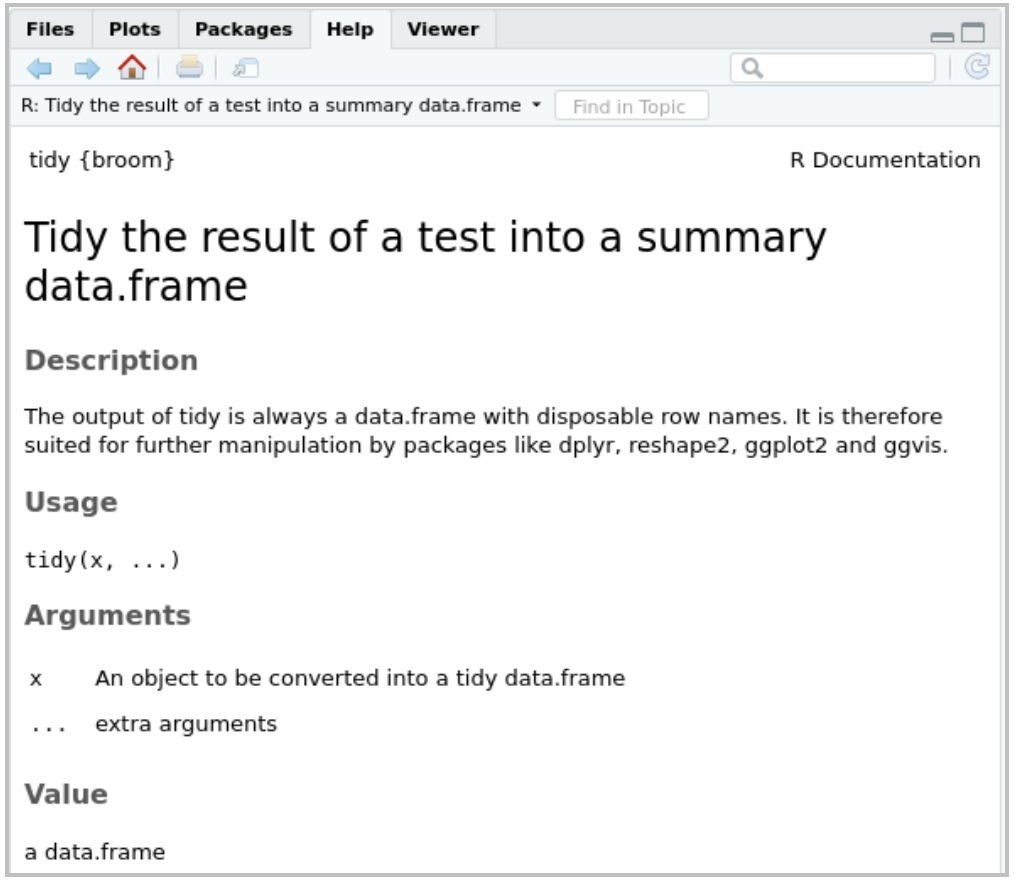
- Note that even though the object is of `dist` class, we still have the values that show the **similarity** scores (i.e. `1 - distance`)

Term similarity matrix: tidying it up

```
?tidy

tidy(x,    #<- an object to be converted
     ...)
```

`tidy` is a function that converts a statistical object like `dist` into a clean data frame with 3 columns:

1. `item1`: the 1st item in the pair of observations from the `dist` object
2. `item2`: the 2nd item in the pair of observations from the `dist` object
3. `distance`: the distance (or similarity, as in our case) value between the two



Files | Plots | Packages | Help | Viewer

R: Tidy the result of a test into a summary data.frame Find in Topic

tidy {broom} R Documentation

Tidy the result of a test into a summary data.frame

Description

The output of tidy is always a data.frame with disposable row names. It is therefore suited for further manipulation by packages like dplyr, reshape2, ggplot2 and ggvis.

Usage

```
tidy(x, ...)
```

Arguments

x	An object to be converted into a tidy data.frame
...	extra arguments

Value

a data.frame

Tidy dist object

```
# Tidy takes an object of class `dist` and
# converts it to a tidy data frame with 3 columns.
wiki_term_sim_df = tidy(wiki_term_sim_half)
head(wiki_term_sim_df)
```

```
# A tibble: 6 x 3
  item1   item2   distance
  <fct>   <fct>      <dbl>
1 also    algebra    0.345
2 analysi algebra    0.148
3 analyt  algebra    0.14
4 appli   algebra    0.653
5 area    algebra    0
6 associ  algebra    0.1
```

```
# Rename columns.
colnames(wiki_term_sim_df) = c("term1", "term2", "cosine_sim")
head(wiki_term_sim_df)
```

```
# A tibble: 6 x 3
  term1   term2   cosine_sim
  <fct>   <fct>        <dbl>
1 also    algebra      0.345
2 analysi algebra      0.148
3 analyt  algebra      0.14
4 appli   algebra      0.653
5 area    algebra      0
6 associ  algebra      0.1
```

Arrange term pairs by similarity scores

```
# Sort entries by decreasing similarity.
wiki_term_sim_df = arrange(wiki_term_sim_df, desc(cosine_sim))
head(wiki_term_sim_df)
```

```
# A tibble: 6 x 3
  term1    term2  cosine_sim
  <fct>    <fct>       <dbl>
1 interest area            1
2 various  area            1
3 equival  associ          1
4 languag  associ          1
5 written  associ          1
6 emerg    biolog          1
```

```
tail(wiki_term_sim_df)
```

```
# A tibble: 6 x 3
  term1   term2   cosine_sim
  <fct>   <fct>        <dbl>
1 vector  trend            0
2 vector  usual            0
3 vector  various          0
4 written various          0
5 visual  vector           0
6 within  vector           0
```

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	
Compare terms by trimming cosine similarity data and building a network graph	
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

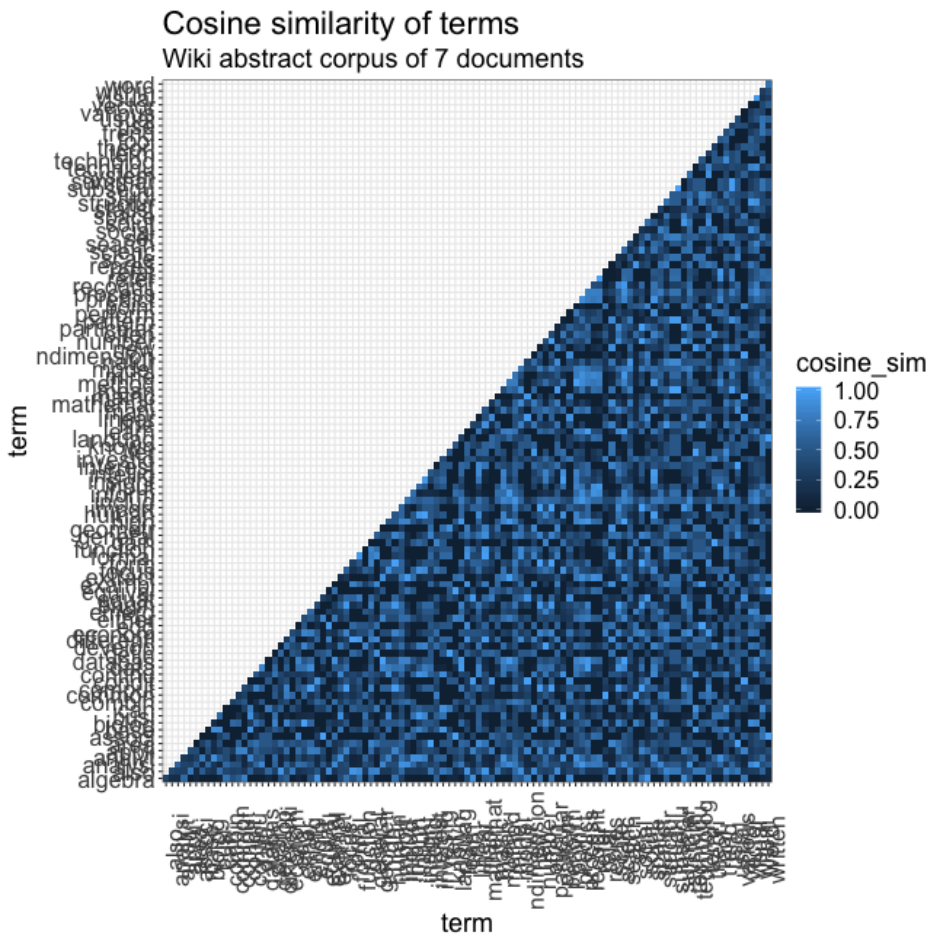
Prepare for plotting: save ggplot theme

```
# Save our custom `ggplot` theme to a variable.
my_ggtheme = theme_bw() +
  theme(axis.title = element_text(size = 18),
        axis.text = element_text(size = 16),
        legend.text = element_text(size = 16),
        legend.title = element_text(size = 18),
        plot.title = element_text(size = 22),
        plot.subtitle = element_text(size = 18))
```

Visualizing term similarity: heatmap

```
# Make a heatmap of term similarity.
wiki_term_heatmap =
  ggplot(data = wiki_term_sim_df,#<- set data
       aes(x = term1,            #<- map term1 to x
           y = term2,            #<- map term2 to y
           fill = cosine_sim)) + #<- map scores
  geom_tile() +                  #<- makes a heatmap
  my_ggtheme +                   #<- add theme
  theme(axis.text.x =
          element_text(angle = 90)) + #<- rotate labs
  xlab("term") +                 #<- set x-axis lab
  ylab("term") +                 #<- set y-axis lab
  labs(title = "Cosine similarity of terms",
    subtitle = "Wiki abstract corpus of 7 documents")
```

```
# View the heatmap.
wiki_term_heatmap
```



- Even though we have trimmed our corpus term dictionary to only `102` terms, it's still quite a bit to view on a single graph in this format

Knowledge check 1

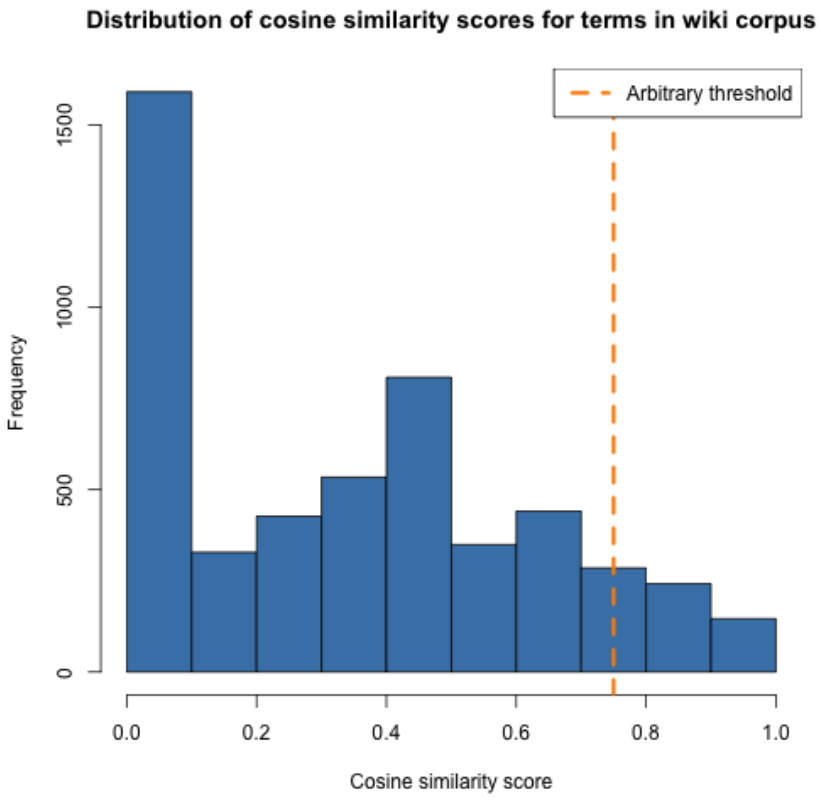
Exercise 1

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Cosine similarity score distribution

- Notice that a lot of scores are `0`
- Most scores are under `0.5`
- You can **create an arbitrary threshold and subset your term similarity pairs** using it
- We will do that in order to **view the terms and their similarity as a network graph**



Visualizing term similarity: network graphs

Every network graph consists of 2 main components:

1. **Nodes**
2. **Edges**

It's common practice to visualize connections and similarity of text data as a **network of terms or documents**, where they represent **nodes** and the **similarity scores** play a role of **edges and their values**.

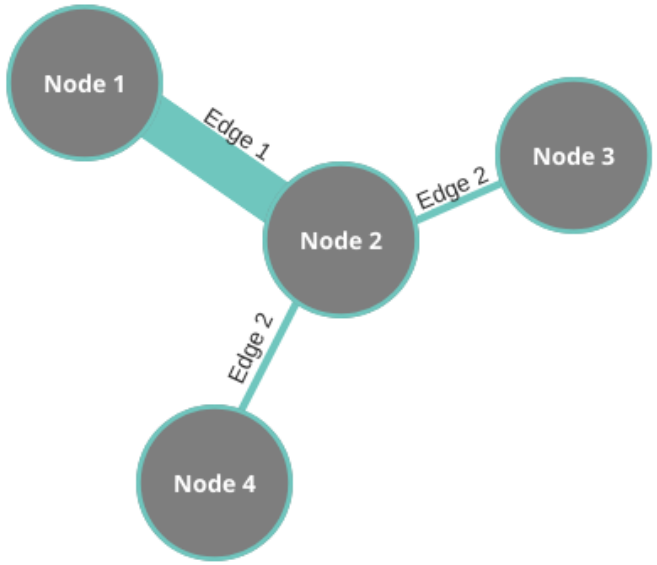
Edges

from	to	value
Node 1	Node 2	0.99
Node 2	Node 3	0.1
Node 2	Node 4	0.1

Nodes

id
Node 1
Node 2
Node 3
Node 4

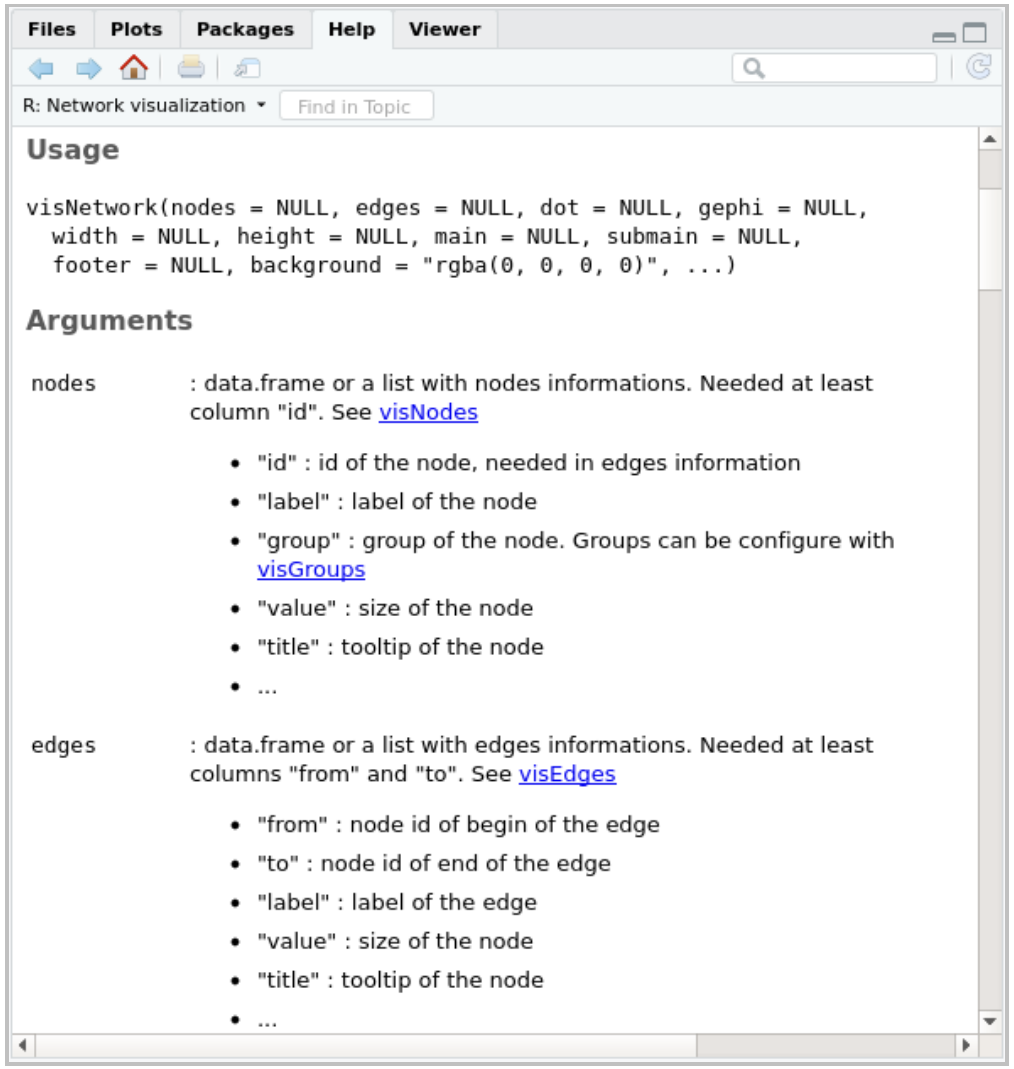
Network graph



Building network graphs with visNetwork

```
?visNetwork

visNetwork(nodes,   #<- data frame with nodes
           edges,   #<- data frame with edges
           ...)
```

- Before we build the graph, we need to create `node` and `edges` data frames



Files | Plots | Packages | Help | Viewer

R: Network visualization | Find in Topic

Usage

```
visNetwork(nodes = NULL, edges = NULL, dot = NULL, gephi = NULL,
  width = NULL, height = NULL, main = NULL, submain = NULL,
  footer = NULL, background = "rgba(0, 0, 0, 0)", ...)
```

Arguments

nodes : data.frame or a list with nodes informations. Needed at least column "id". See visNodes

- "id" : id of the node, needed in edges information
- "label" : label of the node
- "group" : group of the node. Groups can be configure with visGroups
- "value" : size of the node
- "title" : tooltip of the node
- ...

edges : data.frame or a list with edges informations. Needed at least columns "from" and "to". See visEdges

- "from" : node id of begin of the edge
- "to" : node id of end of the edge
- "label" : label of the edge
- "value" : size of the node
- "title" : tooltip of the node
- ...

Make node data frame

- The `node` data frame **must contain at least 1 column**
  - `id`: a vector of all unique `nodes`, from and to which the edges will be drawn
- Since our `nodes` are terms from the similarity matrix, column or row names of the similarity matrix will be our `node` ids

```
# Create a data frame of unique nodes.
wiki_term_nodes = data.frame(id = colnames(wiki_term_sim), #<- node data frame must an `id` column
                             stringsAsFactors = FALSE)
head(wiki_term_nodes, 5)
```

```
       id
1 algebra
2    also
3 analysi
4  analyt
5   appli
```

Make edges data frame

- The `edges` data frame **must contain at least 3 columns**
  - `from`: a vector of node ids from which the edge it to be drawn
  - `to`: a vector of node ids to which the edge it to be drawn, and
  - `value`: a numeric vector that determines the thickness / weight of the edge
- Since our `edges` are cosine similarity scores, the `from` and `to` columns will be the first 2 columns of `wiki_term_sim_df` and the `value` column will be the `cosine_sim` column from the same data frame

```
# Create a data frame of edges.
# Keep only those scores that are > 0.75.
wiki_term_edges = wiki_term_sim_df[wiki_term_sim_df$cosine_sim > 0.75, ]
head(wiki_term_edges, 5)
```

```
# A tibble: 5 x 3
  term1    term2  cosine_sim
  <fct>    <fct>       <dbl>
1 interest area            1
2 various  area            1
3 equival  associ          1
4 languag  associ          1
5 written  associ          1
```

Preparing term similarity data for network graph

- Rename the columns to match network graph syntax

```
# Each data with edges must have at least
# these three columns: `from`, `to`, and `value`.
colnames(wiki_term_edges) = c("from", "to", "value")
head(wiki_term_edges, 5)
```

```
# A tibble: 5 x 3
  from     to     value
  <fct>    <fct>  <dbl>
1 interest area       1
2 various  area       1
3 equival  associ     1
4 languag  associ     1
5 written  associ     1
```

Visualizing term similarity: network graph

- `visNetwork` is an **interactive network graph** function with a variety of options
- We will use the **two main arguments** in the `visNetwork` function (i.e. `nodes` and `edges`)
- Add **two additional options** through `visOptions`:
  - `highlightNearest`: when you click or select a `node`, this highlights all other `nodes` directly connected to the selected one
  - `nodesIdSelection`: creates a little dropdown menu in the top left corner to select a `node` by its id as an alternative to clicking on it
- Similar to working with `highcharter`, we will **use pipe operator `%>%` to append options** to the main graph

```
# Construct network visualization.
wiki_term_sim_network = visNetwork(wiki_term_nodes,            #<- set nodes
                                   wiki_term_edges) %>%        #<- set edges
  visOptions(highlightNearest = TRUE,  #<- highlight nearest when clicking on a node
             nodesIdSelection = TRUE)  #<- add an id node selection menu
```

Visualizing term similarity: network graph

```
# View interactive network graph.
wiki_term_sim_network
```

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Measuring similarity of documents: compute

- To measure the similarity of `documents` in our corpus, we pass `DTM` to a `CosineSim` function
- `wiki_TDM` has `102 terms` and `7 documents`, the similarity matrix made from `DTM` should be `7 x 7`

```
# To get a DTM matrix, we can either construct it with `DocumentTermMatix` function or
# simply transpose the TDM matrix we already have.
wiki_DTM_matrix = t(wiki_TDM_matrix)

# Compute cosine similarity on the matrix.
wiki_doc_sim = CosineSim(wiki_DTM_matrix)

# Take a look at the structure of the matrix.
str(wiki_doc_sim)
```

```
 num [1:7, 1:7] 1 0.6939 0.1179 0.0558 0.0175 ...
 - attr(*, "dimnames")=List of 2
  ..$ Docs: chr [1:7] "1.txt" "2.txt" "3.txt" "4.txt" ...
  ..$ Docs: chr [1:7] "1.txt" "2.txt" "3.txt" "4.txt" ...
```

Measuring similarity of documents: tidy up

```
# Change column and row names of similarity matrix
# to headings of the corpus.
colnames(wiki_doc_sim) = names(wiki_corpus_clean)
rownames(wiki_doc_sim) = names(wiki_corpus_clean)

# Tidy up the distance object.
wiki_doc_sim_df = tidy(as.dist(wiki_doc_sim))
head(wiki_doc_sim_df, 5)
```

```
# A tibble: 5 x 3
  item1 item2 distance
  <fct> <fct>    <dbl>
1 2.txt 1.txt  0.694
2 3.txt 1.txt  0.118
3 4.txt 1.txt  0.0558
4 5.txt 1.txt  0.0175
5 6.txt 1.txt  0.00787
```

```
# Rename columns of the data freme.
colnames(wiki_doc_sim_df) = c("doc1", "doc2", "cosine_sim")
head(wiki_doc_sim_df, 5)
```

```
# A tibble: 5 x 3
  doc1  doc2  cosine_sim
  <fct> <fct>      <dbl>
1 2.txt 1.txt    0.694
2 3.txt 1.txt    0.118
3 4.txt 1.txt    0.0558
4 5.txt 1.txt    0.0175
```

Measuring similarity of documents: sort

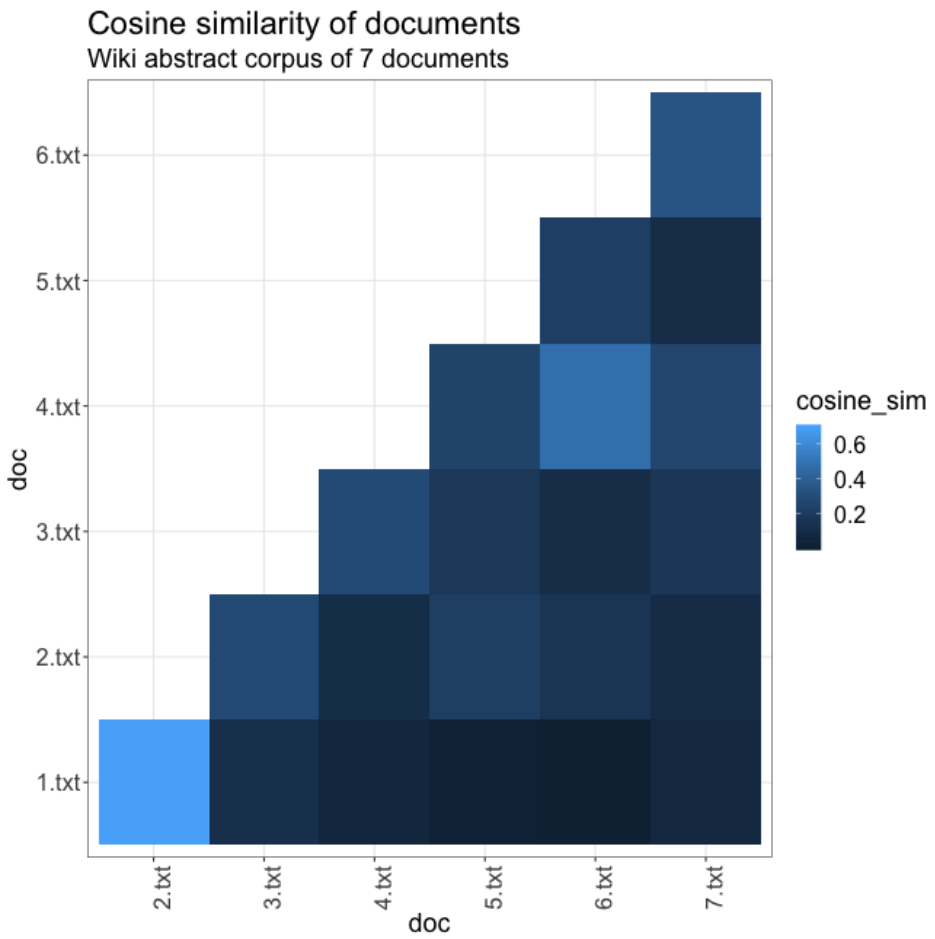
```
# Sort rows by descending similarity scores.
wiki_doc_sim_df = arrange(wiki_doc_sim_df, desc(cosine_sim))
head(wiki_doc_sim_df)
```

```
# A tibble: 6 x 3
  doc1  doc2  cosine_sim
  <fct> <fct>      <dbl>
1 2.txt 1.txt      0.694
2 6.txt 4.txt      0.476
3 7.txt 6.txt      0.340
4 4.txt 3.txt      0.290
5 3.txt 2.txt      0.281
6 7.txt 4.txt      0.255
```

Measuring similarity of documents: plot

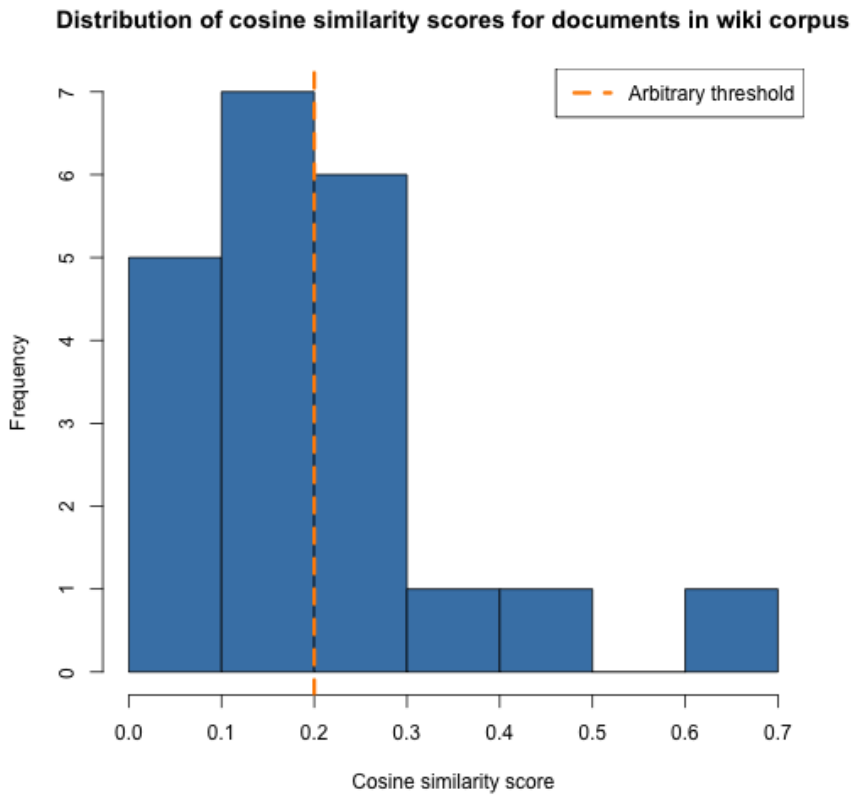
```
# Create a document similarity heatmap.
wiki_doc_heatmap =
  ggplot(data = wiki_doc_sim_df,
         aes(x = doc1,
             y = doc2,
             fill = cosine_sim)) +
  geom_tile() +
  my_ggtheme +
  theme(axis.text.x = element_text(angle = 90)) +
  xlab("doc") +
  ylab("doc") +
  labs(title = "Cosine similarity of documents",
    subtitle = "Wiki abstract corpus of 7 documents")
```

```
# View the heatmap of doc similarity.
wiki_doc_heatmap
```



Cosine similarity score distribution: documents

- Notice that a lot of scores are `0`
- Most scores are under `0.2`
- You can create an arbitrary threshold and subset your term similarity pairs using it
- Let's set our threshold to `0.2`



Create nodes and edges data frames

```
# Create a data frame of unique nodes.
wiki_doc_nodes = data.frame(id = colnames(wiki_doc_sim),
                            stringsAsFactors = FALSE)
head(wiki_doc_nodes, 5)
```

```
     id
1 1.txt
2 2.txt
3 3.txt
4 4.txt
5 5.txt
```

```
# Create a data frame of edges.
# Keep only those scores that are > 0.75.
wiki_doc_edges = wiki_doc_sim_df[wiki_doc_sim_df$cosine_sim > 0.2, ]
head(wiki_doc_edges, 5)
```

```
# A tibble: 5 x 3
  doc1  doc2  cosine_sim
  <fct> <fct>      <dbl>
1 2.txt 1.txt      0.694
2 6.txt 4.txt      0.476
3 7.txt 6.txt      0.340
4 4.txt 3.txt      0.290
5 3.txt 2.txt      0.281
```

Adjust edges data

- Rename the columns to match network graph syntax

```
# Each data with edges must have at least
# these three columns: `from`, `to`, and `value`.
colnames(wiki_doc_edges) = c("from", "to", "value")
head(wiki_doc_edges, 5)
```

```
# A tibble: 5 x 3
  from  to    value
  <fct> <fct> <dbl>
1 2.txt 1.txt 0.694
2 6.txt 4.txt 0.476
3 7.txt 6.txt 0.340
4 4.txt 3.txt 0.290
5 3.txt 2.txt 0.281
```

Visualize document similarity: network graph

```
# Construct network visualization.
wiki_doc_sim_network = visNetwork(wiki_doc_nodes,           #<- set nodes
                                  wiki_doc_edges) %>%       #<- set edges
  visOptions(highlightNearest = TRUE,  #<- highlight nearest when clicking on a node
             nodesIdSelection = TRUE)  #<- add an id node selection menu
```

Visualizing document similarity: network graph

```
# View interactive network graph.
wiki_doc_sim_network
```

Knowledge check 2

Exercise 2

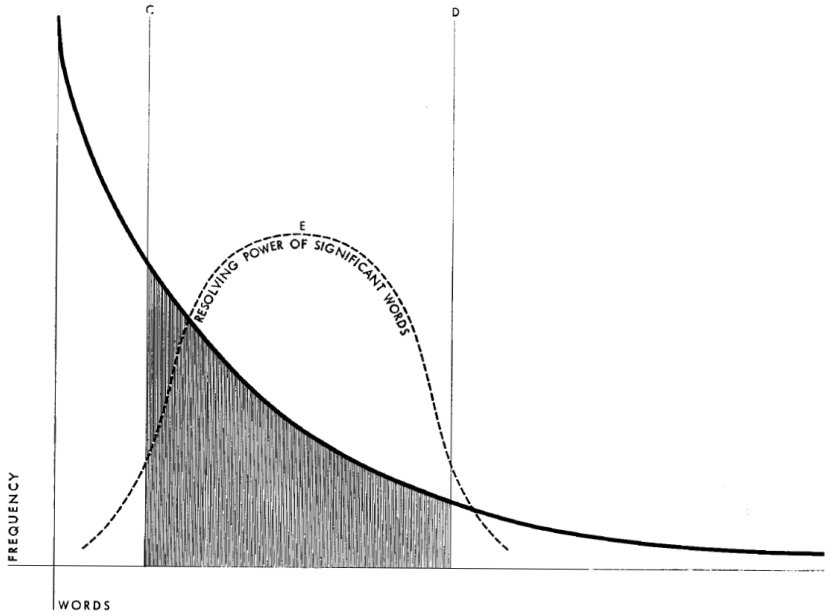
Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Weighting matrices with TF-IDF

- Raw frequency counts work, but not too well. They give disproportionately big weights to words that appear often in many/all documents
- The **terms that make a group of documents stand out** from another group usually **appear frequently, but in few documents**
- Recall the graph that shows how the actual power of significance of words within a corpus follows a bell-shaped curve
- We need to **weigh** our `TDM` or `DTM` in order **to get closer to that shape**
- We will use a simple and effective weighting technique called `TF-IDF`, which stands for **Term Frequency** - **Inverse Document Frequency**

Figure 1 **Word-frequency diagram.**
Abscissa represents individual words arranged in order of frequency.



Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts",*

**Presented at IRE National Convention, New York, March 24, 1958.*

Published in IBM JOURNAL APRIL 1958

TF

1. Stands for `Term Frequency`
2. Is a simple **count of 1 term in 1 document** (that's what we currently have in our `TDM` and `DTM`)

	count	me
doc 1	0	3
doc 2	1	1

- **TF(doc1, count)** = `0`
- **TF(doc1, me)** = `3`
- **TF(doc2, count)** = `1`
- **TF(doc2, me)** = `1`

IDF

1. Stands for `Inverse Document Frequency`
2. Is a *log* of a simple **ratio** of the **total number of documents** towards the **count of documents that contain that term**

	count	me
doc 1	0	3
doc 2	1	1

- **IDF(count)** = `log10(`**2**`/`**1**`) = log10(2) = 0.30103`
- **IDF(me)** = `log10(`**2**`/`**2**`) = log10(1) = 0`

TF-IDF math

TF-IDF

1. Stands for `Term Frequency-Inverse Document Frequency`
2. Is a simple **product** of **TF** and **IDF**

When computed for each entry of a `DTM` or a `TDM`, the matrix is *weighted* with `TF-IDF` instead of simple `TF`!

- **TF(doc1, count)** `X` **IDF(count)** `= 0*0.30103 = 0`
- **TF(doc1, me)** `X` **IDF(me)** `= 3*0 = 0`
- **TF(doc2, count)** `X` **IDF(count)** `= 1*0.30103 = 0.30103`
- **TF(doc2, me)** `X` **IDF(me)** `= 1*0 = 0`

	count	me
doc 1	0	0
doc 2	0.30103	0

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	✔
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Weighting matrices with TF-IDF

- We will make use of the `control` argument of the `TermDocumentMatrix` or `DocumentTermMatrix`
- `control` takes a named list with control functions available in `tm` package
- Among other things, it allows to set `weighting` of the terms

```
# Weighting TDM with TF-IDF.
wiki_DTM_weighted = DocumentTermMatrix(wiki_corpus_clean,
                                       control = list(weighting = weightTfIdf))
wiki_DTM_weighted
```

```
<<DocumentTermMatrix (documents: 7, terms: 466)>>
Non-/sparse entries: 629/2633
Sparsity           : 81%
Maximal term length: 17
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	✔
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	✔
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

Remove sparse terms from weighted DTM

```
# Remove sparse terms from the TDM.
wiki_DTM_weighted = removeSparseTerms(wiki_DTM_weighted,
                                      sparse = 0.75)
```

```
# Inspect the DTM object.
inspect(wiki_DTM_weighted)
```

```
Non-/sparse entries: 265/449
Sparsity           : 63%
Maximal term length: 10
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
         Docs
Terms          1.txt      2.txt      3.txt      4.txt      5.txt       6.txt       7.txt
  algebra 0.02126300 0.10285760 0.00000000 0.00000000 0.00000000 0.000000000 0.000000000
  busi    0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.022313024 0.144588394
  data    0.00000000 0.00000000 0.00000000 0.03175045 0.00000000 0.135821380 0.019558279
  equat   0.02876217 0.03975260 0.00459546 0.00000000 0.00000000 0.000000000 0.000000000
  linear  0.14884099 0.16163337 0.00000000 0.00000000 0.00000000 0.000000000 0.000000000
  mine    0.00000000 0.00000000 0.00000000 0.07041643 0.00000000 0.033469536 0.000000000
  scienc  0.00000000 0.01969158 0.00000000 0.00000000 0.01699695 0.049836724 0.006458839
  social  0.00000000 0.00993815 0.00000000 0.00000000 0.09007102 0.007545632 0.000000000
  space   0.01438109 0.05962890 0.00919092 0.00000000 0.00000000 0.000000000 0.000000000
  vector  0.12757799 0.07346971 0.00000000 0.00000000 0.00000000 0.000000000 0.000000000
```

Compute cosine similarity for weighted DTM

```
# Save the DTM object as a matrix.
wiki_DTM_weighted_matrix = as.matrix(wiki_DTM_weighted)

# Compute cosine similarity of documents in wiki corpus.
wiki_weighted_doc_sim = CosineSim(wiki_DTM_weighted_matrix)

# Rename columns and rows of the similarity matrix.
colnames(wiki_weighted_doc_sim) = meta(wiki_corpus_clean)$heading
rownames(wiki_weighted_doc_sim) = meta(wiki_corpus_clean)$heading
```

```
View(wiki_weighted_doc_sim)
```

wiki_weighted_doc_sim

Filter

	Eigenvalues & eigenvectors	Linear algebra	Fractal geometry	Text mining	Social network analysis	Data science	Business analytics
Eigenvalues & eigenvectors	1.000000000	0.75212269	0.08794378	0.06473540	0.007304692	0.005489412	0.02425901
Linear algebra	0.752122691	1.00000000	0.15091600	0.03120593	0.124321502	0.103212170	0.02105562
Fractal geometry	0.087943783	0.15091600	1.00000000	0.21665863	0.100604530	0.090573765	0.09445073
Text mining	0.064735398	0.03120593	0.21665863	1.00000000	0.087952035	0.452402571	0.13479468
Social network analysis	0.007304692	0.12432150	0.10060453	0.08795203	1.000000000	0.129226783	0.04913348
Data science	0.005489412	0.10321217	0.09057377	0.45240257	0.129226783	1.000000000	0.30507273
Business analytics	0.024259012	0.02105562	0.09445073	0.13479468	0.049133480	0.305072725	1.00000000

Showing 1 to 7 of 7 entries

Tidy up weighted similarity scores

```
# Tidy up the matrix to convert it to a data frame.
wiki_weighted_doc_sim_df = tidy(as.dist(wiki_weighted_doc_sim))

# Rename columns.
colnames(wiki_weighted_doc_sim_df) = c("doc1", "doc2", "cosine_sim")
head(wiki_weighted_doc_sim_df)
```

```
# A tibble: 6 x 3
   doc1  doc2 cosine_sim
  <int> <int>      <dbl>
1     2     1    0.752
2     3     1    0.0879
3     4     1    0.0647
4     5     1    0.00730
5     6     1    0.00549
6     7     1    0.0243
```

```
# Sort rows by scores in decreasing order.
wiki_weighted_doc_sim_df = arrange(wiki_weighted_doc_sim_df,
                                   desc(cosine_sim))
```

Compare doc similarity scores: look at data

```
head(wiki_doc_sim_df, 10)            #<- weighted with TF (i.e. original raw counts)
```

```
# A tibble: 10 x 3
   doc1  doc2  cosine_sim
   <fct> <fct>      <dbl>
 1 2.txt 1.txt      0.694
 2 6.txt 4.txt      0.476
 3 7.txt 6.txt      0.340
 4 4.txt 3.txt      0.290
 5 3.txt 2.txt      0.281
 6 7.txt 4.txt      0.255
 7 5.txt 4.txt      0.245
 8 6.txt 5.txt      0.219
 9 5.txt 2.txt      0.216
10 5.txt 3.txt      0.176
```

```
head(wiki_weighted_doc_sim_df, 10)   #<- weighted with TF-IDF
```

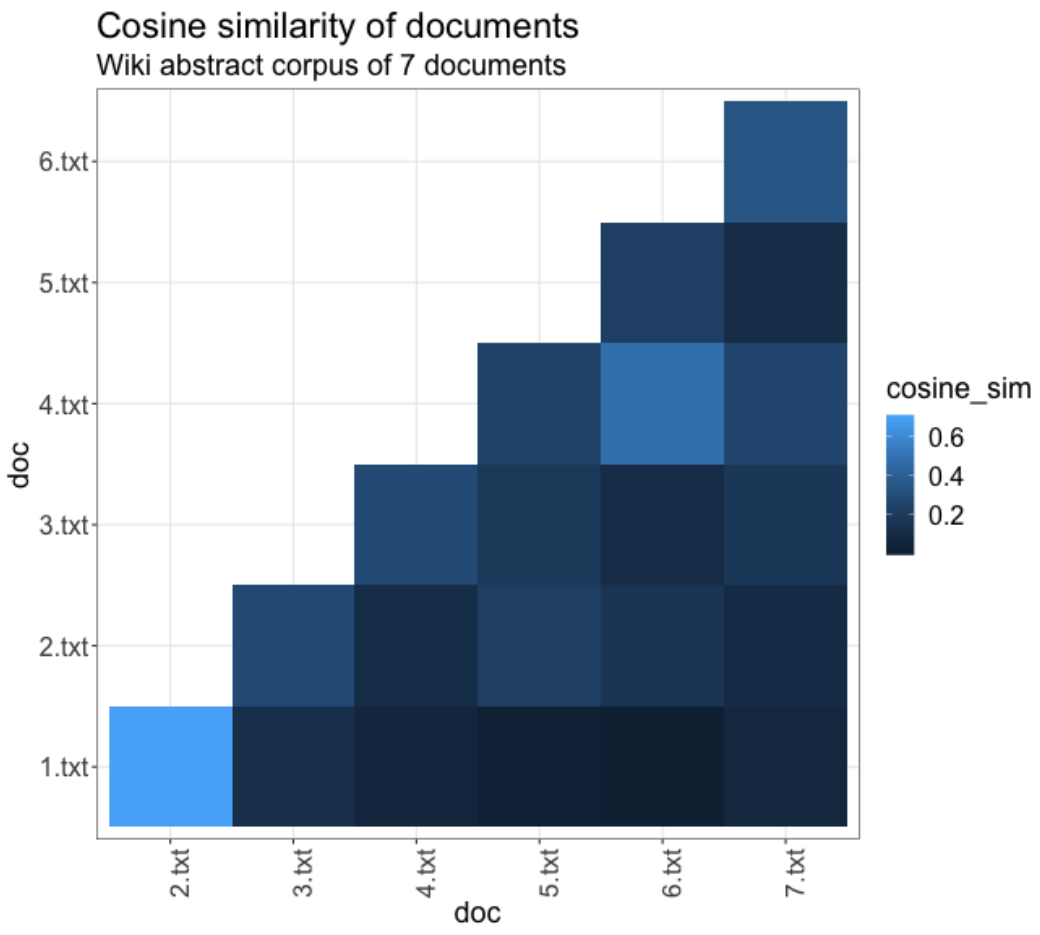
```
# A tibble: 10 x 3
    doc1  doc2 cosine_sim
   <int> <int>      <dbl>
 1     2     1      0.752
 2     6     4      0.452
 3     7     6      0.305
 4     4     3      0.217
 5     3     2      0.151
 6     7     4      0.135
 7     6     5      0.129
 8     5     2      0.124
```

Compare doc similarity scores: create a plot

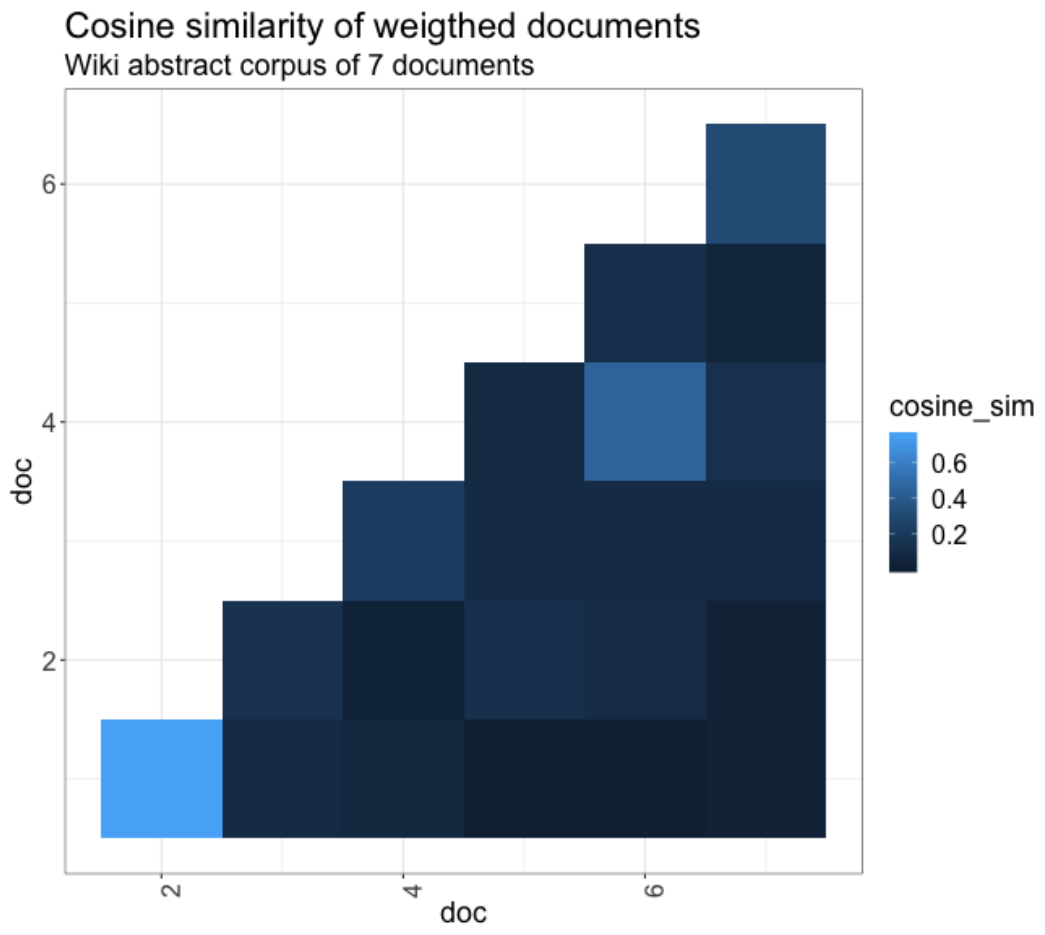
```
# Create a heatmap of the cosine similarity scores
# for weighted document data.
wiki_weighted_doc_heatmap =
  ggplot(data = wiki_weighted_doc_sim_df,
       aes(x = doc1,
           y = doc2,
           fill = cosine_sim)) +
  geom_tile() +
  my_ggtheme +
  theme(axis.text.x = element_text(angle = 90)) +
  xlab("doc") +
  ylab("doc") +
  labs(title = "Cosine similarity of weigthed documents",
       subtitle = "Wiki abstract corpus of 7 documents")
```

Compare doc similarity scores: visualize

```
# Weighted with TF (i.e. original raw counts).
wiki_doc_heatmap
```




```
# Weighted with TF-IDF
wiki_weighted_doc_heatmap
```



Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	✔
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	✔
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	✔
Compute arXiv corpus document similarity scores following prescribed steps	
Build heatmap and network graphs to compare the documents	

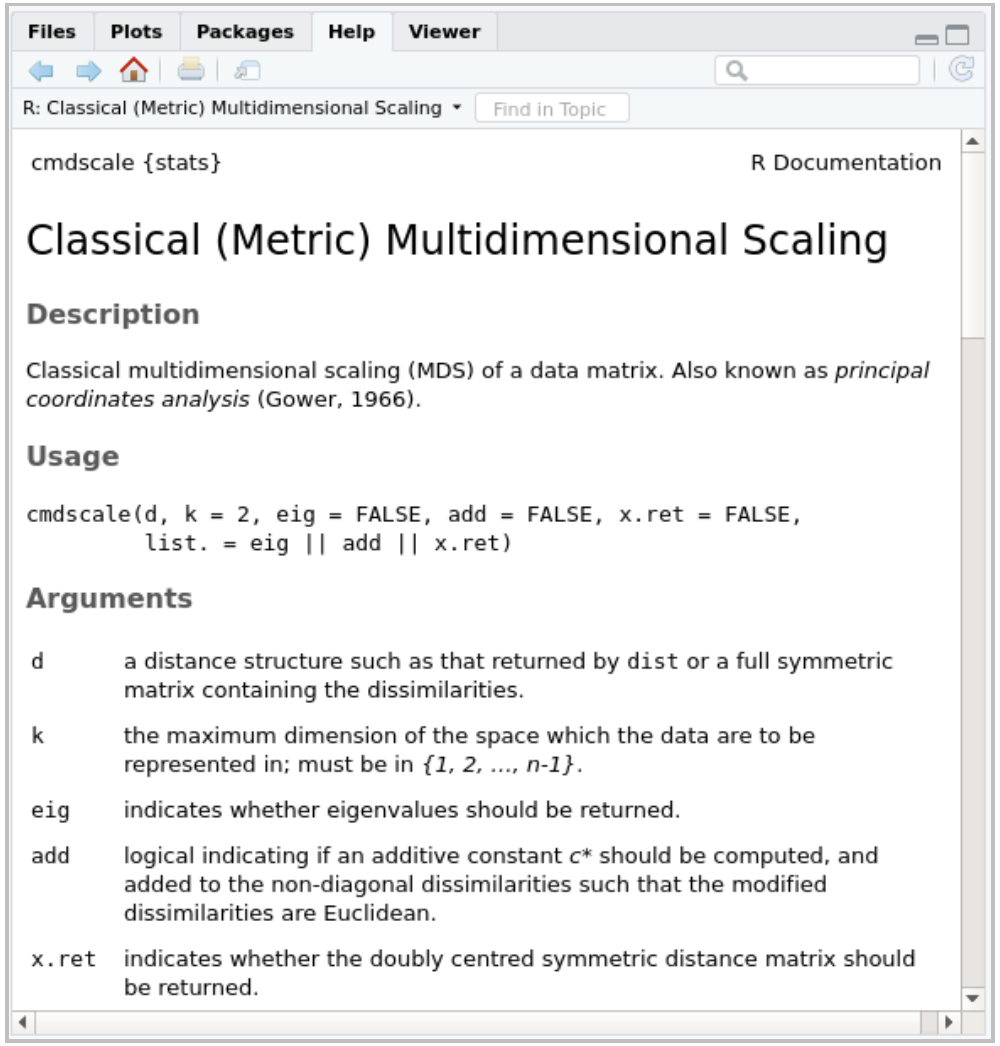
Projecting text data onto x-y plane: cmdscale

- Instead of computing **how close the documents are** to each other, we will **need to know how far away they are**
- We need to **compute distance**, not similarity between them
  - Recall: `Distance = 1 - Similarity`
- We need to **scale the data down from multiple to 2 dimensions**
  - `cmdscale` from the base `stats` function does just that, it projects multi-dimensional data onto a `k`-dimensional plane
  - By default `k = 2`, but you can select a different `k`

Projecting 3D+ data onto x-y plane: cmdscale

```
?cmdscale

cmdscale(x,      #<- dist object
         k = 2, #<- optional # of dimensions
         ...)
```

- `k` must be a number between 1 and one dimension fewer than the current number of variables in data!
- Other arguments are available for fine tuning, but we are only going to use `cmdscale` with its defaults in this class.



Files | Plots | Packages | Help | Viewer

R: Classical (Metric) Multidimensional Scaling Find in Topic

cmdscale {stats} R Documentation

Classical (Metric) Multidimensional Scaling

Description

Classical multidimensional scaling (MDS) of a data matrix. Also known as *principal coordinates analysis* (Gower, 1966).

Usage

```
cmdscale(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE,
         list. = eig || add || x.ret)
```

Arguments

d	a distance structure such as that returned by `dist` or a full symmetric matrix containing the dissimilarities.
k	the maximum dimension of the space which the data are to be represented in; must be in *{1, 2, ..., n-1}*.
eig	indicates whether eigenvalues should be returned.
add	logical indicating if an additive constant *c** should be computed, and added to the non-diagonal dissimilarities such that the modified dissimilarities are Euclidean.
x.ret	indicates whether the doubly centred symmetric distance matrix should be returned.

Convert similarity to distance

```
# Compute cosine distance between wiki docs.
wiki_weighted_doc_dist = 1 - wiki_weighted_doc_sim
```

```
View(wiki_weighted_doc_dist)
```

wiki_weighted_doc_dist

Filter

	Eigenvalues & eigenvectors	Linear algebra	Fractal geometry	Text mining	Social network analysis	Data science	Business analytics
Eigenvalues & eigenvectors	0.0000000	2.478773e-01	9.120562e-01	0.9352646	9.926953e-01	9.945106e-01	9.757410e-01
Linear algebra	0.2478773	1.110223e-16	8.490840e-01	0.9687941	8.756785e-01	8.967878e-01	9.789444e-01
Fractal geometry	0.9120562	8.490840e-01	1.110223e-16	0.7833414	8.993955e-01	9.094262e-01	9.055493e-01
Text mining	0.9352646	9.687941e-01	7.833414e-01	0.0000000	9.120480e-01	5.475974e-01	8.652053e-01
Social network analysis	0.9926953	8.756785e-01	8.993955e-01	0.9120480	4.440892e-16	8.707732e-01	9.508665e-01
Data science	0.9945106	8.967878e-01	9.094262e-01	0.5475974	8.707732e-01	8.881784e-16	6.949273e-01
Business analytics	0.9757410	9.789444e-01	9.055493e-01	0.8652053	9.508665e-01	6.949273e-01	-4.440892e-16

Convert similarity to distance and scale

```
# Save it as a `dist` object.
wiki_weighted_doc_dist = as.dist(wiki_weighted_doc_dist)

# Scale the distance matrix to a 2-dimensional space.
wiki_docs = cmdscale(wiki_weighted_doc_dist)
wiki_docs
```

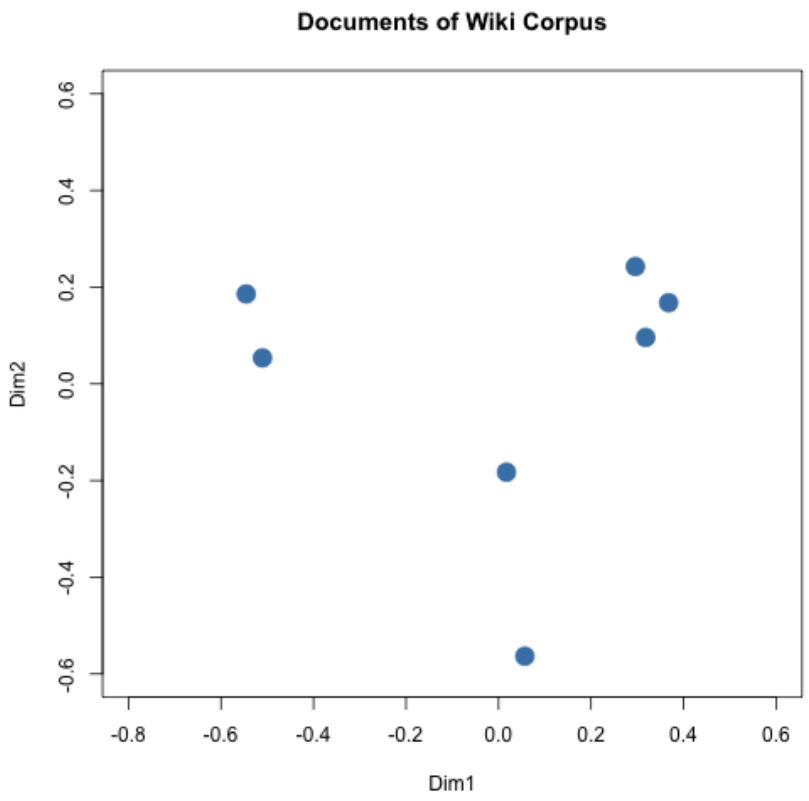
```
            [,1]        [,2]
[1,] -0.54546559  0.18596058
[2,] -0.51027431  0.05363847
[3,]  0.01722141 -0.18279173
[4,]  0.31806863  0.09602056
[5,]  0.05693181 -0.56328030
[6,]  0.36765371  0.16774862
[7,]  0.29586434  0.24270379
```

- We have now a 2D dataset with the 1st column being the `x` coordinate and the 2nd column being the `y` coordinate on our `x-y` plane!

Plotting documents onto an x-y plane

```
# Plot documents on x-y plane.
plot(wiki_docs,                        #<- x-y coordinates
     pch = 19,                         #<- symbol type
     col = "steelblue",                #<- color
     cex = 2,                          #<- size
     xlab = "Dim1",                    #<- x-axis label
     ylab = "Dim2",                    #<- y-axis label
xlim = c(-0.6, 0.6),                   #<- arbitrary x-axis limits
ylim = c(-0.6, 0.6),                   #<- arbitrary y-axis limits
main = "Documents of Wiki Corpus") #<- plot title
text(wiki_docs[, 1] + 0.05,  #<- point labels x-coordinates shifted slightly
     wiki_docs[, 2] + 0.02,  #<- point labels y-coordinates shifted slightly
     rownames(wiki_docs))    #<- text labels for points
```

Documents on an x-y plane



- Notice that our wiki articles do have patterns in similarity
- We can see the graphic summary of what we've already seen through cosine similarity data and plots:
  - **Eigenvalues & eigenvectors** article is very close in meaning to **Linear algebra** article
  - **Business analytics**, **Text mining** and **Data science** articles seem to share a lot of commonalities as well and form a cluster of their own

Introducing the arXiv corpus

https://arxiv.org/help/general

Search

Cornell University Library

We gratefully acknowledge support from the Simons Foundation and member institutions

arXiv.org > help

Search or Article ID | All fields

(Help | Advanced search)

Help Table of Contents | Search arXiv Help

General Information About arXiv

Started in August 1991, arXiv.org (formerly xxx.lanl.gov) is a highly-automated electronic archive and distribution server for research articles. Covered areas include physics, mathematics, computer science, nonlinear sciences, quantitative biology, quantitative finance, statistics, electrical engineering and systems science, and economics. arXiv is maintained and operated by the Cornell University Library with guidance from the arXiv Scientific Advisory Board and the arXiv Member Advisory Board, and with the help of numerous subject moderators.

Users can retrieve papers from arXiv via the web interface. Registered authors may use our web interface to submit their articles to arXiv. Authors can also update their submissions if they choose, though previous versions remain available.

Listings of newly submitted articles in areas of interest are available via the web interface, via RSS feeds, and by subscription to automatic email alerts.

Disclaimer: Papers will be entered in the listings in order of receipt on an impartial basis and appearance of a paper is not intended in any way to convey tacit approval of its assumptions, methods, or conclusions by any agent (electronic, mechanical, or other). We reserve the right to reject any inappropriate submissions.

This site should not be used to distribute non-technical information (such as news or information about political causes of potential special interest to the academic community). Submission of an abstract without an accompanying paper will be rejected outright.

If you have questions about arXiv, please browse our help pages and the arXiv primer, or contact the administrators directly.

Where did the data come from?

https://arxiv.org/help/api/index

Search

Cornell University Library

We gratefully acknowledge support from the Simons Foundation and member institutions

arXiv.org > help > api

Search or Article ID | All fields

(Help | Advanced search)

Help Table of Contents | Search arXiv Help

arXiv API

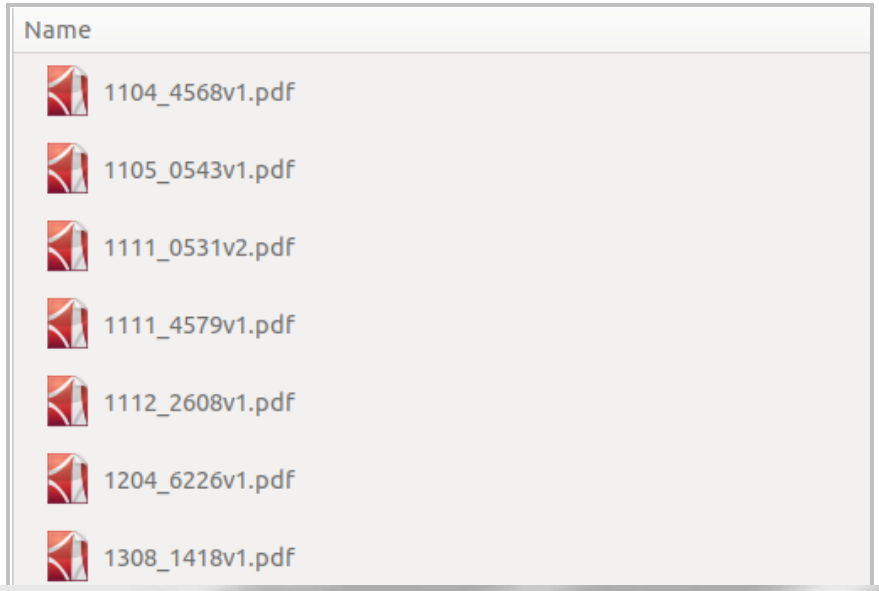
This is the home site of the arXiv API. The goal of the API is to allow application developers access to all of the arXiv data, search and linking facilities with an easy-to-use programmatic interface. This page provides links to developer documentation, and gives instructions for how to join the mailing list and contact other developers and maintainers.

For more information about the arXiv API, please see our arxiv-api group or look at the API FAQ.

- API News
- About the arXiv API
- Quickstart
- Using the arXiv API
- arXiv API documentation
- Community

What did we scrape?

- 36 articles from ***https://arxiv.org***
- Keywords we used for search: `"epidemiology"` and `"virus"`
- Format of original files: `PDF`
- We pre-saved the original articles in the `arxiv_corpus` folder inside of your `data` directory



Understanding the Origins of a Pandemic Virus

Carlos Xavier Hernández[1], Joseph Chan[1], Hossein Khiabanian[1, 2], Raul Rabadan[1, 2*]

[1] Center for Computational Biology and Bioinformatics,

[2] Department of Biomedical Informatics,

Columbia University College of Physicians and Surgeons,

New York, NY, United States

* Corresponding author's email: rabadan@dbmi.columbia.edu

Abstract

Understanding the origin of infectious diseases provides scientifically based rationales for implementing public health measures that may help to avoid or mitigate future epidemics. The recent ancestors of a pandemic virus provide invaluable information about the set of minimal genomic alterations that transformed a zoonotic agent into a full human pandemic. Since the first confirmed cases of the H1N1 pandemic virus in the spring of 2009 [1,2], several hypotheses about the strain's origins have been proposed. However, how, where, and when it first infected humans is still far from clear. The only way to piece together this epidemiological puzzle relies on the collective effort of the international scientific community to increase genomic sequencing of influenza isolates, especially ones collected in the months prior to the origin of the pandemic.

How did we do it?

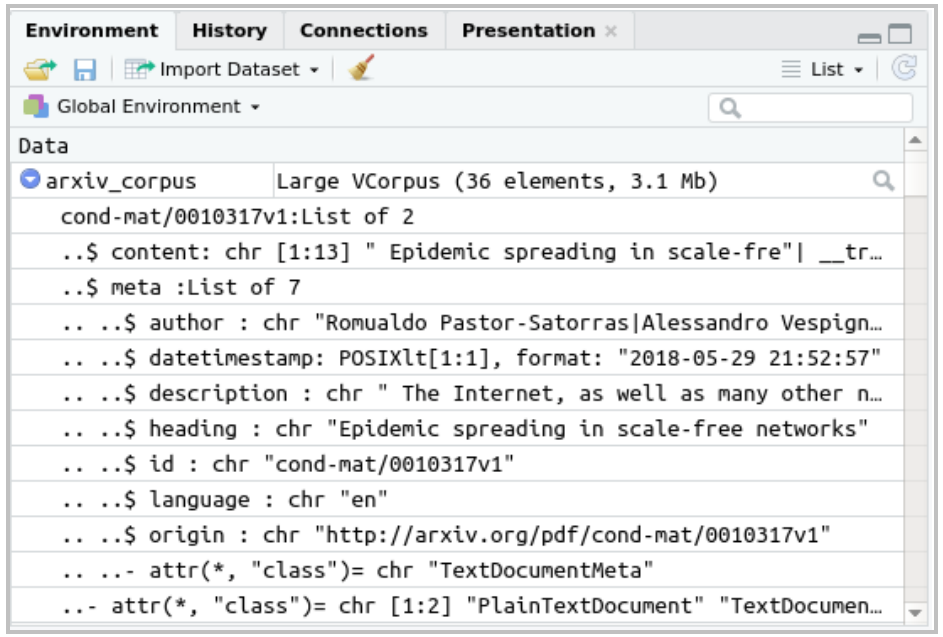
- Libraries we used:
  - `fulltext`: a wrapper around multiple packages and APIs to scrape data from popular scientific article repositories
  - `pdftools`: a package for reading `PDF` into R
  - `tm`: to create a corpus with appropriate metadata
- We are not going to go through the details of scraping and converting the articles, but you can find the code in your supplementary materials and welcome to study it on your own

Load arXiv corpus

- Let's load the `tm` package and the pre-saved `arxiv_corpus.RData` into our environment

```
# Set working directory to `data_dir`.
setwd(data_dir)

# Load corpus scraped from arXiv.org.
load("arxiv_corpus.RData")

# Load clean corpus scraped from arXiv.org.
load("arxiv_corpus_clean.RData")
```

- Take a look at your environment, you should be able to inspect it in environment window



Explore arXiv corpus

```
# Take a look at a short corpus summary.
arxiv_corpus
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 16
Content:  documents: 36
```

```
# View global corpus metadata.
View(meta(arxiv_corpus))
```

meta(arxiv_corpus)

Filter

	id	submitted	updated	title	abstract	authors
1	cond-mat/0010317v1	2000-10-20 16:57:05	2000-10-20 16:57:05	Epidemic spreading in scale-free networks	The Internet, as well as many other networks, has a...	Romualdo Pastor-Satorras|Alessandro Ve
2	q-bio/0601035v3	2006-01-22 23:13:25	2006-10-03 19:00:13	A Comparative Analysis of Influenza Vaccination Pro...	The threat of avian influenza and the 2004-2005 infl...	Shweta Bansal|Babak Pourbohloul|Laure
3	1104.4568v1	2011-04-23 15:58:43	2011-04-23 15:58:43	Understanding the Origins of a Pandemic Virus	Understanding the origin of infectious diseases prov...	Carlos Xavier Hernandez|Joseph Chan|Ho
4	1105.0543v1	2011-05-03 10:55:55	2011-05-03 10:55:55	HIV dynamics and natural history studies: Joint mod...	Hepatitis C virus (HCV) coinfection has become one ...	Li Su|Joseph W. Hogan
5	1111.0531v2	2011-11-02 15:17:14	2011-11-06 03:26:25	Virus Dynamics on Starlike Graphs	The field of epidemiology has presented fascinating ...	Thealexa Becker|Alexander Greaves-Tun
6	1111.4579v1	2011-11-19 19:41:26	2011-11-19 19:41:26	Canalization of the evolutionary trajectory of the hu...	Since its emergence in 1968, influenza A (H3N2) ha...	Trevor Bedford|Andrew Rambaut|Merced
7	1112.2608v1	2011-12-12 16:26:51	2011-12-12 16:26:51	Rohlin Distance and the Evolution of Influenza A vir...	The evolution of the hemagglutinin amino acids seq...	Raffaella Burioni|Riccardo Scalco|Mario C
8	1204.6226v1	2012-04-27 14:25:16	2012-04-27 14:25:16	A Novel Sequence-Based Antigenic Distance Measur...	H1N1 influenza causes substantial seasonal illness ...	Keyao Pan|Krystina C. Subieta|Michael W
9	1308.1418v1	2013-08-06 20:40:52	2013-08-06 20:40:52	A Latent Social Approach to YouTube Popularity Pred...	Current works on Information Centric Networking as...	Amandianeze O Nwana|Salman Avestime
10	1308.5140v2	2013-08-23 14:41:24	2014-03-21 13:27:15	Simultaneous reconstruction of evolutionary history ...	The evolution of RNA viruses such as HIV, Hepatitis ...	Denise Kühnert|Tanja Stadler|Timothy G.
11	1311.1870v1	2013-11-08 02:55:38	2013-11-08 02:55:38	Mapping of the Influenza-A Hemagglutinin Serotype...	Analyses and visualizations by the ISSCOR method ...	Jan P. Radomski|Piotr P. Slonimski|Włodzi

Showing 1 to 12 of 36 entries

Inspect document level metadata

```
# Take a look at the individual document's metadata.
meta(arxiv_corpus[[1]])
```

```
  author       : Romualdo Pastor-Satorras|Alessandro Vespignani
  datetimestamp: 2018-05-29 21:52:57
  description  :   The Internet, as well as many other networks, has a very complex connectivity
recently modeled by the class of scale-free networks. This feature, which
appears to be very efficient for a communications network, favors at the same
time the spreading of computer viruses. We analyze real data from computer
virus infections and find the average lifetime and prevalence of viral strains
on the Internet. We define a dynamical model for the spreading of infections on
scale-free networks, finding the absence of an epidemic threshold and its
associated critical behavior. This new epidemiological framework rationalize
data of computer viruses and could help in the understanding of other spreading
phenomena on communication and social networks.

  heading      : Epidemic spreading in scale-free networks
  id           : cond-mat/0010317v1
  language     : en
  origin       : http://arxiv.org/pdf/cond-mat/0010317v1
```

Look at lengths of character strings

```
# Take a look at the number of characters in the first document.
nchar(content(arxiv_corpus[[1]]))
```

```
 [1] 2941 2551 2558 2611 2614 2336 2320 1394 1728 1225 1263  635 1522
```

```
# Each document consists of a vector of character strings.
# Every entry in a vector was a page in the document.
# To view how many pages there are in each document, you can
# use the `length` command.
length(content(arxiv_corpus[[1]]))
```

```
[1] 13
```

Weighted DTM for arXiv articles

- Let's test our document comparison analysis pipeline on `arXiv.org` article corpus
- We will be comparing documents, so we need to construct a weighted `DTM`

```
# Create and weight DTM with TF-IDF for arxiv articles.
arxiv_DTM_weighted = DocumentTermMatrix(arxiv_corpus_clean,
                                        control = list(weighting = weightTfIdf))
arxiv_DTM_weighted
```

```
<<DocumentTermMatrix (documents: 36, terms: 10532)>>
Non-/sparse entries: 30990/348162
Sparsity           : 92%
Maximal term length: 20
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

Remove sparse terms + compute cosine similarity

```
# Remove sparse terms (set threshold at 0.75).
arxiv_DTM_weighted = removeSparseTerms(arxiv_DTM_weighted,
                                       sparse = 0.75)
arxiv_DTM_weighted
```

```
<<DocumentTermMatrix (documents: 36, terms: 778)>>
Non-/sparse entries: 13222/14786
Sparsity           : 53%
Maximal term length: 12
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

```
# Save it as a matrix.
arxiv_DTM_weighted_matrix = as.matrix(arxiv_DTM_weighted)

# Compute cosine similarity.
arxiv_weighted_doc_sim = CosineSim(arxiv_DTM_weighted_matrix)
str(arxiv_weighted_doc_sim)
```

```
 num [1:36, 1:36] 1 0.106 0.0417 0.0671 0.2511 ...
 - attr(*, "dimnames")=List of 2
  ..$ Docs: chr [1:36] "cond-mat/0010317v1" "q-bio/0601035v3" "1104.4568v1" "1105.0543v1" ...
  ..$ Docs: chr [1:36] "cond-mat/0010317v1" "q-bio/0601035v3" "1104.4568v1" "1105.0543v1" ...
```

Tidy up weighted arXiv similarity data

```
# Tidy up the similarity matrix.
arxiv_weighted_doc_sim_df = tidy(as.dist(arxiv_weighted_doc_sim))
head(arxiv_weighted_doc_sim_df)
```

```
# A tibble: 6 x 3
  item1           item2               distance
  <fct>           <fct>                  <dbl>
1 q-bio/0601035v3 cond-mat/0010317v1   0.106
2 1104.4568v1     cond-mat/0010317v1   0.0417
3 1105.0543v1     cond-mat/0010317v1   0.0671
4 1111.0531v2     cond-mat/0010317v1   0.251
5 1111.4579v1     cond-mat/0010317v1   0.0378
6 1112.2608v1     cond-mat/0010317v1   0.108
```

```
# Rename columns of the data frame.
colnames(arxiv_weighted_doc_sim_df) = c("doc1", "doc2", "cosine_sim")
head(arxiv_weighted_doc_sim_df)
```

```
# A tibble: 6 x 3
  doc1            doc2                cosine_sim
  <fct>           <fct>                    <dbl>
1 q-bio/0601035v3 cond-mat/0010317v1     0.106
2 1104.4568v1     cond-mat/0010317v1     0.0417
3 1105.0543v1     cond-mat/0010317v1     0.0671
4 1111.0531v2     cond-mat/0010317v1     0.251
5 1111.4579v1     cond-mat/0010317v1     0.0378
6 1112.2608v1     cond-mat/0010317v1     0.108
```

Sort weighted arXiv similarity data

```
# Sort data by descending similarity.
arxiv_weighted_doc_sim_df = arrange(arxiv_weighted_doc_sim_df, desc(cosine_sim))
head(arxiv_weighted_doc_sim_df)
```

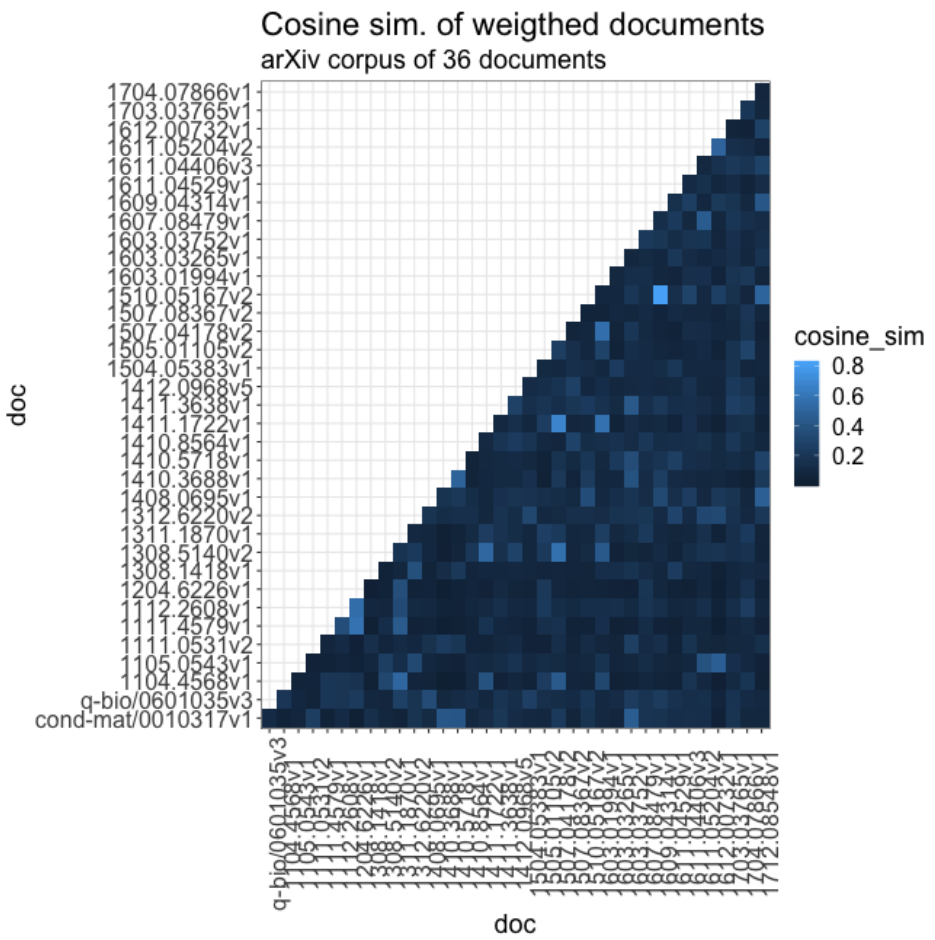
```
# A tibble: 6 x 3
  doc1         doc2         cosine_sim
  <fct>        <fct>             <dbl>
1 1609.04314v1 1510.05167v2      0.812
2 1507.04178v2 1411.1722v1       0.679
3 1204.6226v1  1111.4579v1       0.582
4 1603.01994v1 1411.1722v1       0.575
5 1507.04178v2 1308.5140v2       0.568
6 1603.01994v1 1507.04178v2      0.559
```

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	✔
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	✔
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	✔
Compute arXiv corpus document similarity scores following prescribed steps	✔
Build heatmap and network graphs to compare the documents	

Visualize weighted arXiv similarity data

```
# Plot similarity data on a heatmap.
ggplot(data = arxiv_weighted_doc_sim_df,
       aes(x = doc1,
           y = doc2,
           fill = cosine_sim)) +
  geom_tile() +
  my_ggtheme +
  theme(axis.text.x = element_text(angle = 90)) +
  xlab("doc") +
  ylab("doc") +
  labs(title = "Cosine sim. of weigthed documents",
       subtitle = "arXiv corpus of 36 documents")
```



Cosine distance of arXiv articles: scale

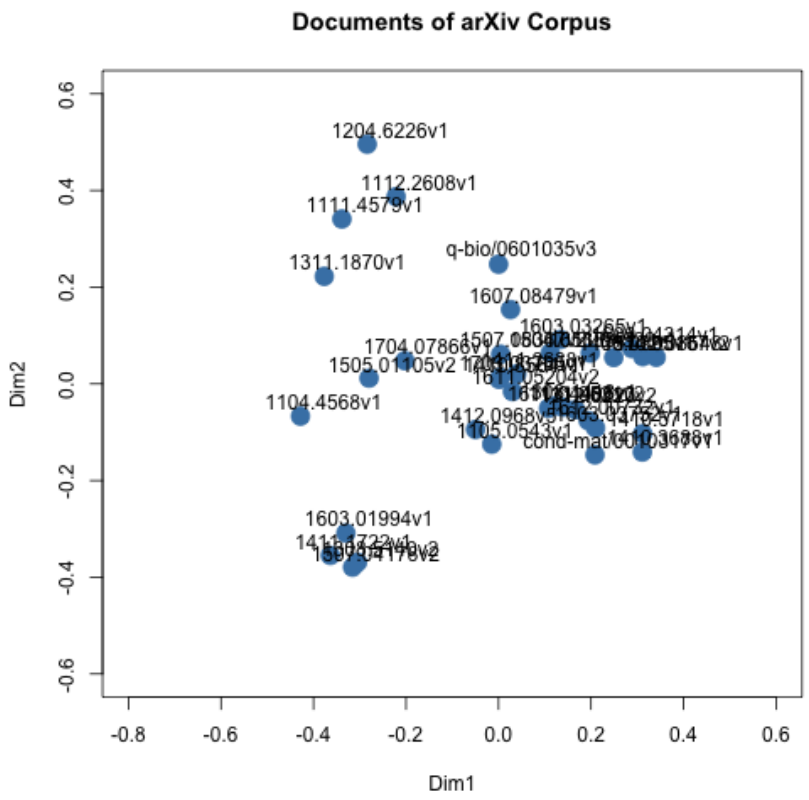
```
# Compute cosine distance and scale the arxiv document data.
arxiv_docs = cmdscale(as.dist(1 - arxiv_weighted_doc_sim))

# Take a look at the result.
head(arxiv_docs)
```

```
                           [,1]        [,2]
cond-mat/0010317v1  0.208455695 -0.14688029
q-bio/0601035v3     0.000154447  0.24736871
1104.4568v1        -0.428245715 -0.06680228
1105.0543v1        -0.014598882 -0.12487565
1111.0531v2         0.139810433 -0.04821035
1111.4579v1        -0.338618501  0.34101709
```

Plotting arXiv documents onto an x-y plane

```
# Plot documents on x-y plane.
plot(arxiv_docs,                          #<- x-y coordinates
     pch = 19,                            #<- symbol type
     col = "steelblue",                   #<- color
     cex = 2,                             #<- size
     xlab = "Dim1",                       #<- x-axis label
     ylab = "Dim2",                       #<- y-axis label
     xlim = c(-0.8, 0.6),                 #<- arbitrary x-axis limits
     ylim = c(-0.6, 0.6),                 #<- arbitrary y-axis limits
     main = "Documents of arXiv Corpus") #<- plot title
text(arxiv_docs[, 1] + 0.05,        #<- point labels x-coordinates shifted slightly
     arxiv_docs[, 2] + 0.03,        #<- point labels y-coordinates shifted slightly
     rownames(arxiv_docs))          #<- text labels for points
```

View arXiv documents on an x-y plane



- Notice that our arXiv articles also have patterns in similarity

Create node and edge data frames for arXiv data

```
# Create a data frame of unique nodes.
arXiv_doc_nodes = data.frame(id = meta(arxiv_corpus_clean)$id,
                             label = meta(arxiv_corpus_clean)$id,    #<- node label
                             title = meta(arxiv_corpus_clean)$title,#<- node tooltip
                             stringsAsFactors = FALSE)
str(arXiv_doc_nodes)
```

```
'data.frame':   36 obs. of  3 variables:
 $ id   : chr  "cond-mat/0010317v1" "q-bio/0601035v3" "1104.4568v1" "1105.0543v1" ...
 $ label: chr  "cond-mat/0010317v1" "q-bio/0601035v3" "1104.4568v1" "1105.0543v1" ...
 $ title: chr  "Epidemic spreading in scale-free networks" "A Comparative Analysis of Influenza
Vaccination Programs" "Understanding the Origins of a Pandemic Virus" "HIV dynamics and natural history
studies: Joint modeling with doubly\n  interval-censored event time and infreq"| __truncated__ ...
```

```
# Create a data frame of edges.
# Keep only those scores that are > 0.75.
arXiv_doc_edges = arxiv_weighted_doc_sim_df[arxiv_weighted_doc_sim_df$cosine_sim > 0.25, ]
str(arXiv_doc_edges)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   63 obs. of  3 variables:
 $ doc1      : Factor w/ 36 levels "cond-mat/0010317v1",..: 29 22 8 25 22 25 8 17 11 36 ...
 $ doc2      : Factor w/ 36 levels "cond-mat/0010317v1",..: 24 17 6 17 10 22 7 10 3 24 ...
 $ cosine_sim: num  0.812 0.679 0.582 0.575 0.568 ...
```

Rename edges columns

- Rename the columns to match network graph syntax

```
# Each data with edges must have at least
# these three columns: `from`, `to`, and `value`.
colnames(arXiv_doc_edges) = c("from", "to", "value")
head(arXiv_doc_edges, 5)
```

```
# A tibble: 5 x 3
  from         to           value
  <fct>        <fct>        <dbl>
1 1609.04314v1 1510.05167v2 0.812
2 1507.04178v2 1411.1722v1  0.679
3 1204.6226v1  1111.4579v1  0.582
4 1603.01994v1 1411.1722v1  0.575
5 1507.04178v2 1308.5140v2  0.568
```

Plot network graph of arXiv documents

```
# Construct network visualization.
arXiv_doc_sim_network = visNetwork(arXiv_doc_nodes,          #<- set nodes
                                   arXiv_doc_edges) %>%      #<- set edges
  visOptions(highlightNearest = TRUE,  #<- highlight nearest when clicking on a node
             nodesIdSelection = TRUE)  #<- add an id node selection menu
```

Visualize arXiv documents in network graph

```
# View interactive network graph.
arXiv_doc_sim_network
```

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Identify the need to reduce corpus sparsity and introduce removeSparseTerms function	✔
Introduce the concept of cosine similarity and distance	✔
Compute term similarity matrix for wiki corpus and transform it into tidy data format	✔
Create wiki corpus term similarity heatmap	✔
Compare terms by trimming cosine similarity data and building a network graph	✔
Compute cosine similarity for wiki corpus documents, transform the data and visualize it it	✔
Identify the need for weighting text frequency data, introduce the concept of TF-IDF weights	✔
Demonstrate weighting with TF-IDF using weightTfIdf control option in tm package	✔
Compute wiki corpus document similarity scores, build visualizations and compare to non-weighted data	✔
Compute arXiv corpus document similarity scores following prescribed steps	✔
Build heatmap and network graphs to compare the documents	✔

Congratulations on completing this module!