# Computational Social Science

## Introduction to Natural Language Processing II

Dr. Thomas Davidson

Rutgers University

February 29, 2024

# Plan

1. Course updates
2. TF-IDF weighting
3. Vector representations of texts
4. Cosine similarity

# Course updates

- Homework 2 grades
- Project proposals due next Wednesday at 5pm
  - Complete quiz on Canvas

# Working with text

### Recap

- Introduction to Natural Language Processing
- Pre-processing texts
  - Tokenization, stemming, stop word removal
- The bag-of-words representation
  - N-grams

# Working with text

## Comparing documents

- The goal of today's lecture is to introduce methods for comparing documents
  - Re-weighting word counts to find distinctive words
  - Representing documents as vectors of word counts
  - Geometric interpretations of document vectors

# Working with text

### Limitations of word counts

- Word counts alone are an imperfect measure for comparing documents
  - Some words occur in most documents, providing little information about the document (recall Zipf's law)
  - Similarly, some words are very rare, providing little generalizable insight
  - We want to find words that help distinguish between documents

# Working with text

### Term-frequency inverse document-frequency (TF-IDF)

- Term-frequency inverse document-frequency (TF-IDF) is a way to weight word counts ("term frequencies") to give higher weights to words that help distinguish between documents
  - Intuition: Adjust word counts to take into account how many documents a word appears in.

# Working with text

### Calculating term-frequency inverse document-frequency (TF-IDF)

- $N$ = number of documents in the corpus
- $tf_{t,d}$ = number of times term $t$ used in document $d$
- $df_t$ = number of documents containing term $t$
- $idf_t = log(\frac{N}{df_t})$ = log of fraction of all documents containing $t$
  - $\frac{N}{df_t}$ is larger for terms occurring in fewer documents
  - The logarithm is used to penalize very high values
  - If a word occurs in all documents $df_t = N$, thus $idf_t = log\frac{N}{N} = log(1) = 0$ .
- We then use these values to calculate $TFIDF_{t,d} = tf_{t,d} * idf_t$

# Working with text

### Loading data

Loading the word frequency objects created last lecture using `tidytext`.

# Working with text

### Computing TF-IDF in `tidytext`

We can easily compute TF-IDF weights using `tidy.text` by using the word-count object we created last lecture. Note the two document example is quite trivial. Many words have IDF scores equal to zero because they occur in both documents.

```
tidy.tfidf <- words %>% bind_tf_idf(word, title, n)
head(tidy.tfidf)
```

```
## # A tibble: 6 x 7
##   title             word          n total      tf   idf tf_idf
##   <chr>             <chr>     <dbl> <dbl>   <dbl> <dbl>  <dbl>
## 1 Elementary Forms totem      1250 78851 0.0159  0.693 0.0110
## 2 Elementary Forms religi      606 78851 0.00769 0     0
## 3 Elementary Forms anim        577 78851 0.00732 0     0
## 4 Elementary Forms religion    572 78851 0.00725 0     0
## 5 Elementary Forms form        542 78851 0.00687 0     0
## 6 Elementary Forms natur       542 78851 0.00687 0     0
```

# Working with text

Take the stem "countri" for example (short for country, country's, countries).

```
## # A tibble: 2 x 7
##   title                word        n total       tf   idf tf_idf
##   <chr>                <chr>   <dbl> <dbl>    <dbl> <dbl>  <dbl>
## 1 Communist Manifesto countri    26  4835 0.00538      0      0
## 2 Elementary Forms    countri    16 78851 0.000203     0      0
```

# Working with text

The term "australia" has a relatively low term frequency but a higher IDF score, since it only occurs in *Elementary Forms*.

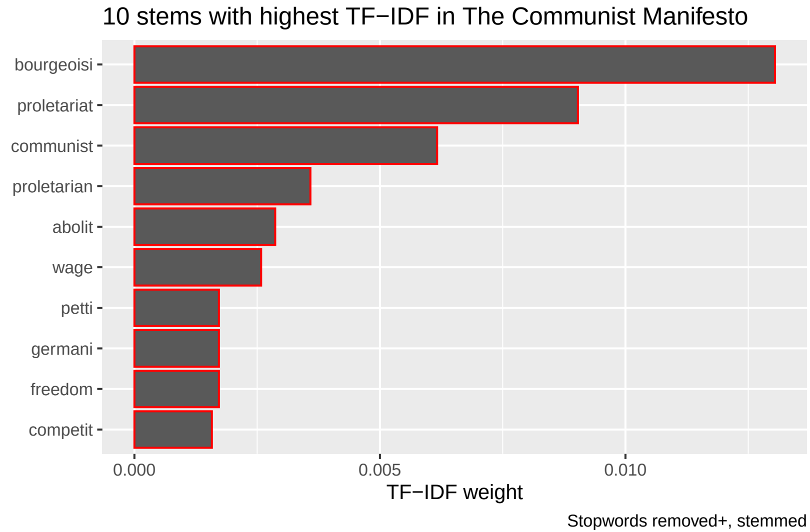
```
## # A tibble: 1 x 7
##   title             word            n total       tf   idf   tf_idf
##   <chr>             <chr>       <dbl> <dbl>    <dbl> <dbl>    <dbl>
## 1 Elementary Forms australia    108 78851 0.00137 0.693 0.000949
```

# Working with text

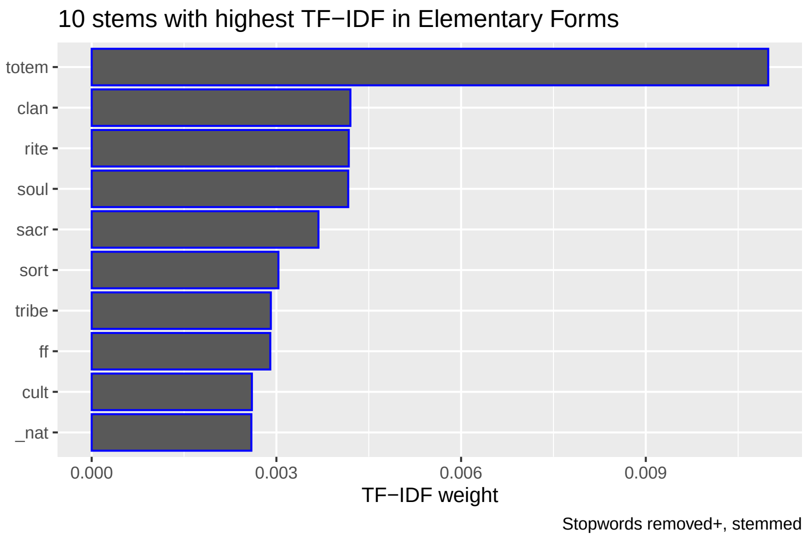
In this case *all* words unique to one document will have the same IDF score, $\sim log(2/1)$.

```
## # A tibble: 6 x 7
##   title              word      n total      tf   idf  tf_idf
##   <chr>              <chr> <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 Elementary Forms totem  1250 78851 0.0159  0.693 0.0110
## 2 Elementary Forms clan    478 78851 0.00606 0.693 0.00420
## 3 Elementary Forms rite    475 78851 0.00602 0.693 0.00418
## 4 Elementary Forms soul    474 78851 0.00601 0.693 0.00417
## 5 Elementary Forms sacr    419 78851 0.00531 0.693 0.00368
## 6 Elementary Forms sort    345 78851 0.00438 0.693 0.00303
```

# Working with text



# Working with text



# Vector representations of texts

### The document-term matrix (DTM)

- A frequently used bag-of-words representation of a text corpus is the *Document-Term Matrix*:
  - Each row* is a document (a unit of text)
  - Each column is a term (word)
  - For a given DTM $X$, each cell $X_{i,j}$ indicates the number of times a term $i$ occurs in document $j$, $tf_{i,j}$.
    - This can be the raw term counts or TF-IDF weighted counts.
- Most cells are empty so it is usually stored as a sparse matrix to conserve memory.

*Sometimes the rows and columns are reversed, resulting in a *Term-Document Matrix* or *TDM*

# Vector representations of texts

### Casting a tidytext object into a DTM

```
X <- texts %>% unnest_tokens(word, text) %>%
    anti_join(stop_words) %>% count(title, word) %>%
    cast_dtm(title, word, n)
print(X)
```

```
## <<DocumentTermMatrix (documents: 2, terms: 11525)>>
## Non-/sparse entries: 12663/10387
## Sparsity           : 45%
## Maximal term length: NA
## Weighting          : term frequency (tf)
```

Note: This matrix is not weighted by TF-IDF, although we could apply the weights if desired.

# Vector representations of texts

## Viewing the DTM

The object created is a class unique to the `tidytext` package. We can inspect this to see what it contains.

```
class(X)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
dim(X)
```

```
## [1]     2 11525
```

```
X$dimnames[1]
```

```
## $Docs
## [1] "Communist Manifesto" "Elementary Forms"
```

```
#X$dimnames[2] # prints all columns as a long list
X$dimnames[[2]][1:50] # first 50 columns
```

```
##  [1] "1"                 "10"                "1830"              "1846"
##  [5] "1847"              "1888"              "18th"              "2"
##  [9] "3"                 "4"                 "5"                 "6"
## [13] "7"                 "8"                 "9"                 "_a"
```

# Vector representations of texts

### Viewing the DTM

The easiest way to see the actual DTM is to cast it to a matrix.

```
Xm <- as.matrix(X)
```

# Vector representations of texts

### Geometric interpretation

- Each text is a vector in N-dimensional space, where N is the total number of unique words (column of the DTM)
- Each word is a vector in D-dimensional space, where D is the number of documents (rows of the DTM)

See https://web.stanford.edu/~jurafsky/slp3/6.pdf for more details on the vector-space model

# Vector representations of texts

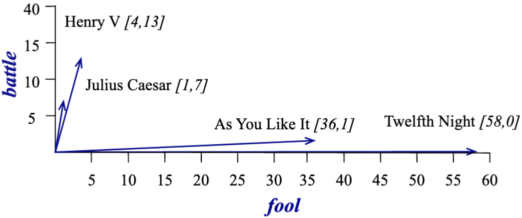
### Document vectors

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 0 | 7 | 13 |
| good | 114 | 80 | 62 | 89 |
| fool | 36 | 58 | 1 | 4 |
| wit | 20 | 15 | 2 | 3 |

This example from Jurafsky and Martin shows a Term-Document Matrix (TDM) pertaining to four key words from four Shakespeare plays. The document vectors are highlighted in red.

# Vector representations of texts

## Document vectors




Here vectors for each play are plotted in two-dimensional space. The y- and x-axes indicate the number of times the words “battle” and “fool” appear in each play. Note how some vectors are closer than others and how they have different lengths.

# Vector representations of texts

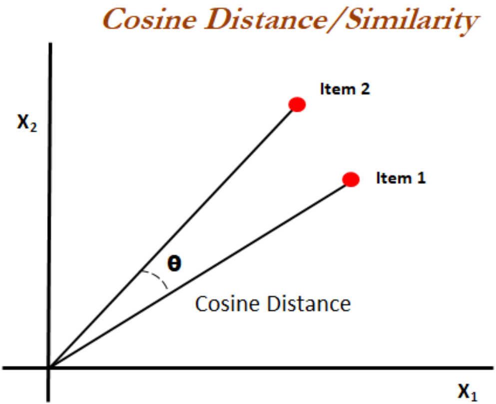
## Word vectors

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 0 | 7 | 13 |
| good | 114 | 80 | 62 | 89 |
| fool | 36 | 58 | 1 | 4 |
| wit | 20 | 15 | 2 | 3 |

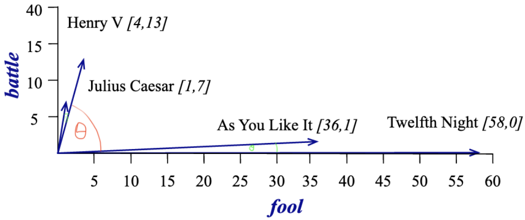
**Figure 6.5** The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

We could also treat the rows of this matrix as vector representations of each word. We will return to this idea next week.

# Cosine similarity



# Cosine similarity



# Cosine similarity

$\vec{u}$ and $\vec{v}$ are vectors representing texts (e.g. rows from a DTM matrix). We can compute the cosine of the angle between these two vectors using the following formula:

$$cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} = \frac{\sum_i \vec{u}_i \vec{v}_i}{\sqrt{\sum_i \vec{u}_i^2} \sqrt{\sum_i \vec{v}_i^2}}$$

The value range from 0 (complete dissimilarity) to 1 (identical), since all values are non-negative.

# Cosine similarity

```r
u <- c(1,2,3,4)
v <- c(0,1,0,1)

sum(u*v) / (sqrt(sum(u^2)) * sqrt(sum(v^2)))
```

```
## [1] 0.7745967
```

```r
# Same result using matrix multiplication
u %*% v / (sqrt(u %*% u) * sqrt(v %*% v))
```

```
##           [,1]
## [1,] 0.7745967
```

# Cosine similarity

## Making a function

```
cosine.sim <- function(u,v) {
  numerator <- u %*% v
  denominator <- sqrt(u %*% u) * sqrt(v %*% v)
  return (numerator/denominator)
}

cosine.sim(u,v)
```

```
##           [,1]
## [1,] 0.7745967
```

# Cosine similarity

### Cosine similarity between Marx and Durkheim

We can use the two columns of the DTM matrix defined above as arguments to the similarity function.

```r
print(cosine.sim(Xm[1,], Xm[2,]))
```

```
##           [,1]
## [1,] 0.6012744
```

# Cosine similarity

### Cosine similarity for a larger corpus

Let's consider another example with a slightly larger corpus of texts.

```
m <- gutenberg_metadata %>%
    filter(author == "Shakespeare, William" & language == "en")
plays <- gutenberg_download(2235:2269)

plays <- plays %>% left_join(m, by = "gutenberg_id") %>%
    filter(gutenberg_id != 2240) # Removing a duplicate
```

# Cosine similarity

### From text to DTM

```
## <<DocumentTermMatrix (documents: 33, terms: 5194)>>
## Non-/sparse entries: 72926/98476
## Sparsity           : 57%
## Maximal term length: 15
## Weighting          : term frequency (tf)
## [1]   33 5194
```

# Cosine similarity

### Extracting TF-IDF matrix

```r
DTMd <- as.matrix(DTM)

# Run line below if using tf-idf weights as
# some columns contain zeros and must be removed
#DTMd <- DTMd[,colSums(DTM) > 0]
```

# Cosine similarity

### Normalizing columns

We can simplify the cosine similarity calculation if we normalize each column by its length (the denominator in the above calculation.)

```
normalize <- function(v) {
  return (v/sqrt(v %*% v))
}

# Normalizing every column in the matrix
for (i in 1:dim(DTMd)[1]) {
  DTMd[i,] <- normalize(DTMd[i,])
}
```

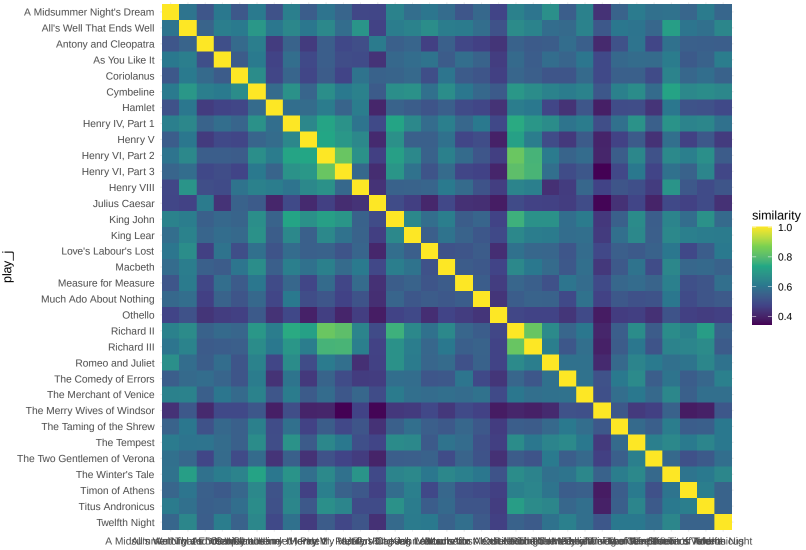
# Cosine similarity

## Calculating cosine similarity using matrix multiplication

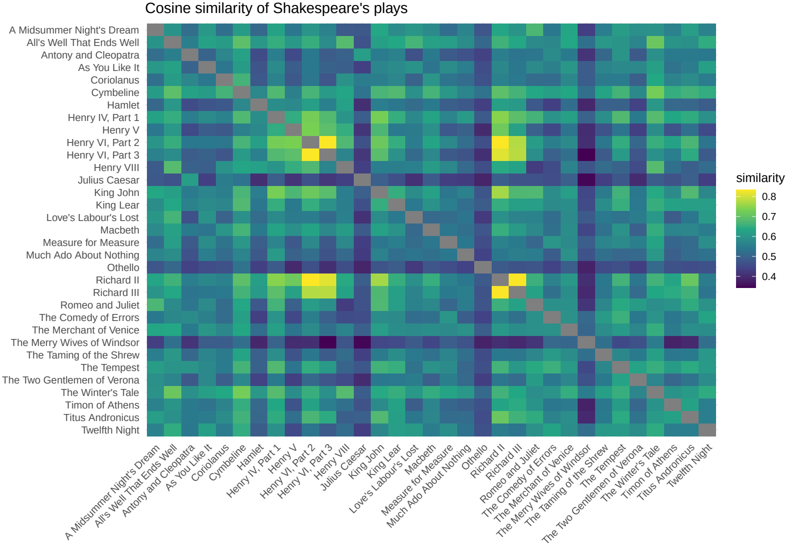
```
sims <- DTMd %*% t(DTMd)
print(sims)
```

```
##                              Docs
## Docs                           A Midsummer Night's Dream
##   A Midsummer Night's Dream                    1.0000000
##   All's Well That Ends Well                    0.5957040
##   Antony and Cleopatra                         0.5197436
##   As You Like It                               0.6053355
##   Coriolanus                                   0.5247506
##   Cymbeline                                    0.6119579
##   Hamlet                                       0.5085226
##   Henry IV, Part 1                             0.6259541
##   Henry V                                      0.5376199
##   Henry VI, Part 2                             0.5960737
##   Henry VI, Part 3                             0.5571196
##   Henry VIII                                   0.4827328
##   Julius Caesar                                0.4800865
```

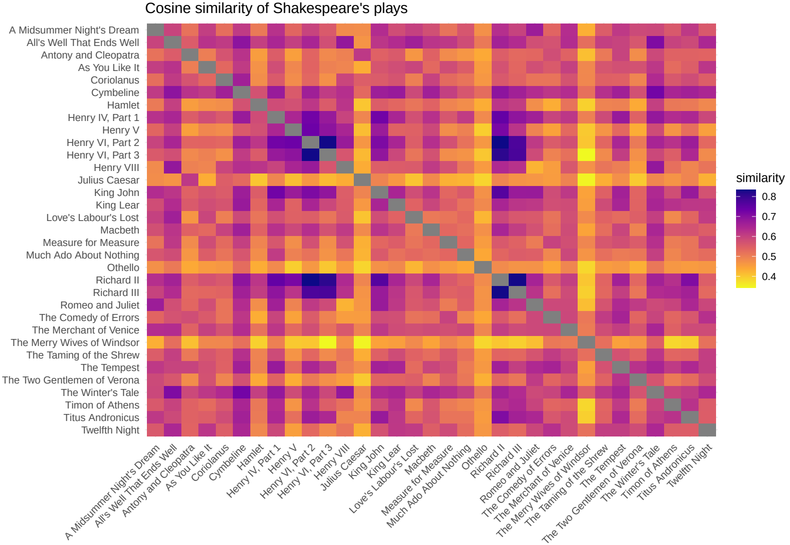
# Cosine similarity



# Cosine similarity



# Cosine similarity



## Next week

- Word embeddings