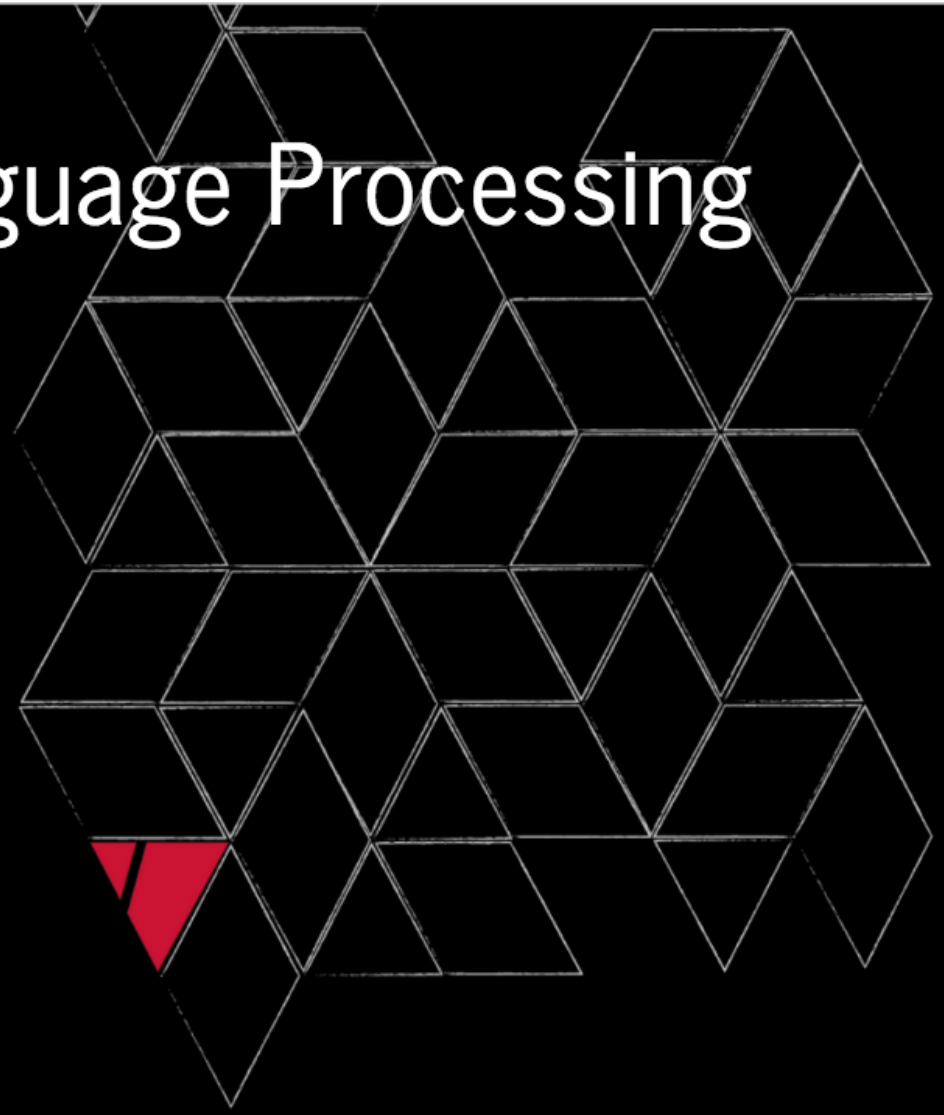
Intro to Natural Language Processing

Dan Acheson

Data Scientist

timshel.



Lots and lots of text

- **100s** of comments on your Facebook page
- **1000s** of emails in your inbox
- **5 Million** articles on Wikipedia
- **500 Million** tweets a day
- **1 Billion** webpages and counting

Doing good with Textual Data

- Combining text and geocodes from Twitter during/after natural disasters
- Finding trends and topics in UNICEF's U-Report
- Analyzing articles to understand the relationships between the UNs Sustainable Development Goals
- Using the Twitter API to understand your community!

Data, data everywhere... but how can we unlock it?

- A vast amount is available, but:
  - It's owned / curated by someone else who wants $$$
  - It's locked up in the form of text
- With a few techniques and some free tools, a vast world of data can be unlocked

So, what DO you do if someone gives you this?!?

```
<body>
    <header>
        <h1>Welcome to the Web Editor!</h1>
        <nav>
            <a href="index.html" class="welcome_tile selected">Welcome</a>
            <a href="hardcore.html" class="hardcore_tile">Get Hardcore</a>
            <a href="tips.html" class="tips_tile">Tips &amp; Tricks</a>
        </nav>
    </header>
    <article>
        <h2>What is the &ldquo;Web&rdquo; in the Web Editor?</h2>
        <p>Well, you&rsquo;re looking at it: a truly versatile, powerful, amazing, n
        <div>
            <h3><em>Click right here</em> for the first bit o&rsquo; magic.</h3>
            <p>Did you see how the corresponding HTML code has been brought into viev
            <p>What about locating a tag in the preview from the source code? Look at
            <img src="images/img_2.png" alt="">
            <p>A word of caution before you call yourself a web expert with this secr
            <p>Now double-click any text in the Preview. See the text-editing tools t
            <h4>Here&rsquo;s how we use this every day:</h4>
            <ul>
                <li>You can find HTML elements fast and easily, no matter how far awa
                <li>Instantly see the effect of any changes you make to HTML or CSS c
                <li>Figure out what styles are applied to that one piece of content</
                <li>Say goodbye to copy/paste nightmares between your development and
            </ul>
```


The New York Times.

AUSTRIA FORMALLY DECLARES WAR ON SERVIA; RUSSIA THREATENS, ALREADY MOVING TROOPS; PEACE OF EUROPE NOW IN KAISER'S HANDS

Timshel @timshel · Apr 7

We're excited to help the #nonprofit community scale social impact

Built In Chicago @BuiltInChicago

How Obama's former CTO gives organizations the tools to build movements: bit.ly/1qwOAFT @timshel

8 12

Timshel @timshel · Apr 6

Thanks @ChicagoInno for the article on how we're using #tech to help #nonprofits activate their communities @jdallke ow.ly/10n16B

7 10

Timshel @timshel · Apr 4

Thanks for the follow @fayfeeney! We look forward to continuing the conversation about #socialgood.

1

Timshel @timshel · Mar 30

Excited to share #NLP techniques at #DoGoodData2016 next month @meansandrew @DataScience_Dan ow.ly/106mlj

4 6

Timshel @timshel · Mar 25

One way that #technology is helping the humanitarian community reach people in vulnerable areas @guardian

Natural Language Processing

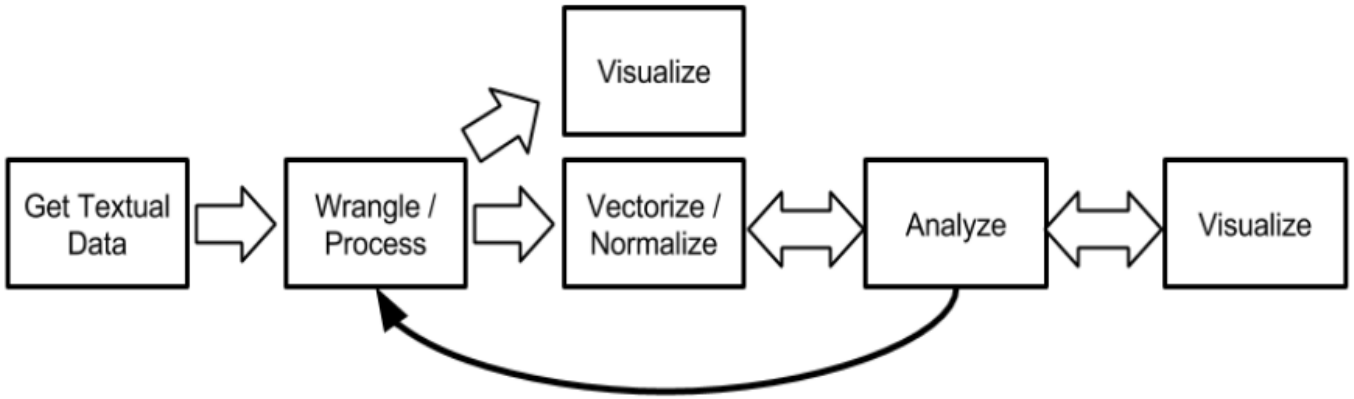
- Bringing order to the chaos

- Techniques, tools and algorithms that will allow for computational approaches to language processing

- Historically comes from the combination of a number of fields

- Computer science
- Computational linguistics
- Artificial Intelligence
- Psychology

Typical NLP Workflow



Today's Tutorial

- **Can be hands on**
- **Can also just follow my slides and look at code later**
- **Get R and Python code here:**

http://bit.ly/1QyDrZF

Today's Tutorial

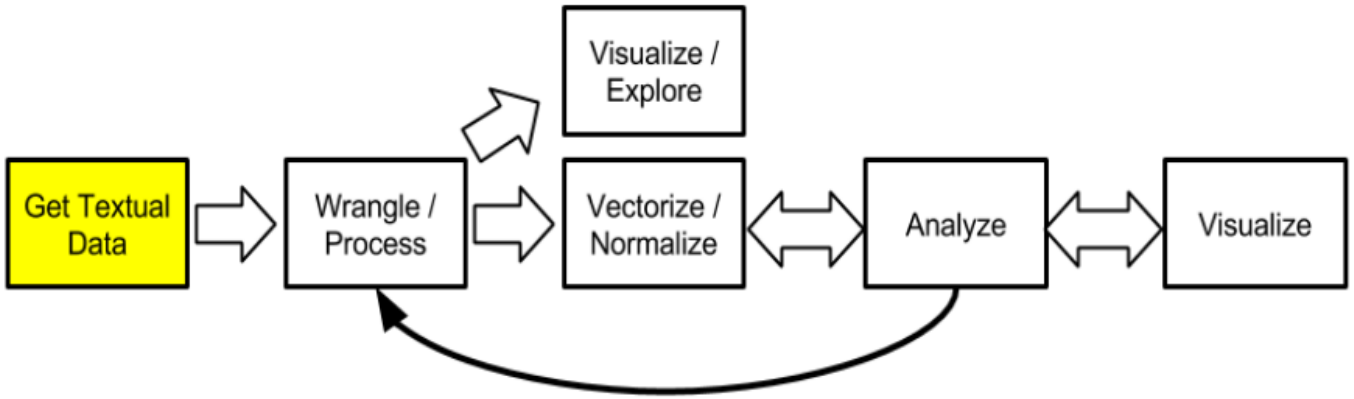
What we'll go through:

- Terminology and data sources
- Text wrangling 101
- Pre-processing for NLP
- The vector-space model of text
- Machine learning

Terminology

- **Natural language:** language produced by humans
- **Text**: the writen form of language
- **String**: computer representation of text (a.k.a., character)
- **Document**: a bunch of text that comes from the same place
  - Can be as large as a whole book, or as small as a single tweet
- **Corpus**: A collection of documents (plural: *corpora*)

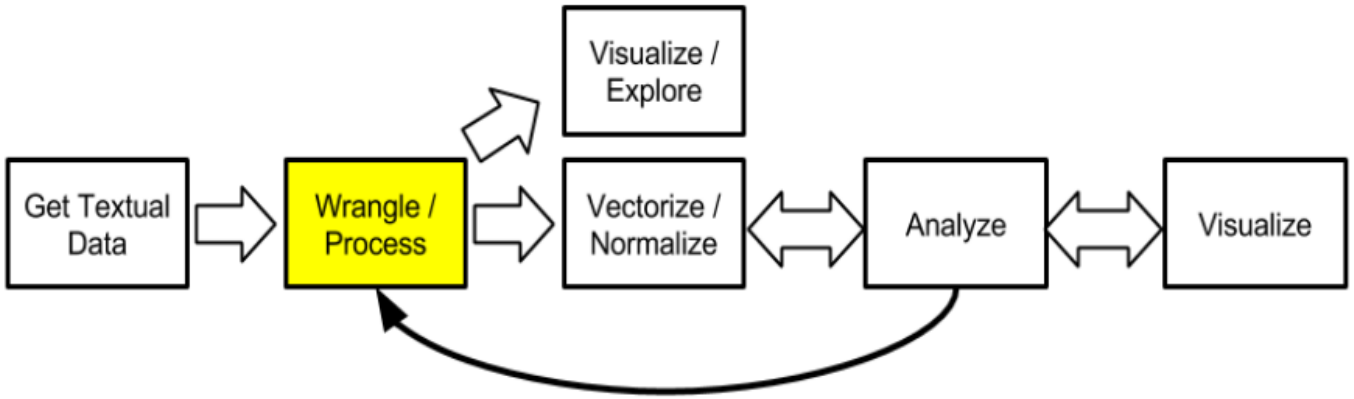
Data Sources for Text



Data Sources for Text

1. APIs from the services you love
   - Twitter API is particularly is useful
2. Web-scraping
3. PDFs / printed text
   - Need optical character recognition (OCR)
4. Digitized collections of books, newspapers, etc.
   - Project Gutenberg
   - New York Times
5. Curated corpora by NLP researchers
6. Wikipedia!

Text Wrangling 101



Text Wrangling 101

Very important skills to learn if you're going to work with text

- Base R has some decent functionality

```
grep gsub substr strsplit tolower/toupper paste/paste0
```

- Some good packages that I'll use below

```
stringr stringi
```

Typical Processing Operations

- making text **`lower`** or **`UPPER`** case
- **`splitting`** text on a delimiter
- **`substituting`** one string for another
- **`searching`** for or **`extracting`** strings
- **`trimming`** text to remove leading a trailing whitespace
- performing the above operations on a list of strings

Typical Processing Operations

We don't have time today to go through all these operations

Examples of each are provided in the R and Python code

One thing we will explore a bit, though, are Regular Expressions

Regular Expressions:

- Regular Expressions (i.e., regexes) are a simple yet powerful language for finding patterns in text
- If you don't know 'em, go out and learn 'em.
- You'll find tons of applications and seriously increase your ability to search, munge and manipulate text.
- For more info, check the following: regex in stringi

Regular Expression Syntax:

Syntax	Example
`[ ]` set membership	`[A-Za-z]` = all letters
`[^ ]` set exclusion	`[^0-9]` = exclude numbers
* repeat pattern 0 or more times	`[a-z]*` = lowercase repeat 1+
+ repeat pattern 1 or more times	`ab+` = "ab" repeated 0+ times
`{1,3}` repeats between 1-3X	`[a-z]{1,3}` = lowercase letters 1-3X
\ escape character to ignore regex syntax (\\ in R)	`\\.com` = treat "." like a period
`.` anything once	`b.b` matches "bob", "bub" but not "barb"
`.*` anything repeated (greedy) **Be Careful!** `.*?` is non-greedy	`b\.*b` matches anything beginning and ending with "b"
^ begins with	`^[A-Z]` = begins with capital letter
$ ends with	`.*ing$` = anything ending with "ing"

Example: Extracting domain names from email addresses

```
myText: Address 1: dan@timshel.com, Address 2:
barney_rubble@flinstones.com
```

Extract email addresses using the `stringr` package

Remember `.*` is *greedy*, so be careful!

Here using `.*?` and escaping the . with `\\.`

```
#NOTE: In R, escape is \\
emails = str_extract_all(myText, "[a-zA-Z0-9_]+@.*?\\.[a-zA-Z]+",
simplify = T)
emails
```

```
     [,1]              [,2]
[1,] "dan@timshel.com" "barney_rubble@flinstones.com"
```

Get Domain Names

Split emails using the "@" symbol

```
email_split <- sapply(emails, function(x) str_split(x, "@"),
USE.NAMES = F)
email_split
```

```
[[1]]
[1] "dan"          "timshel.com"

[[2]]
[1] "barney_rubble"   "flinstones.com"
```

Extract domains

```
domain_names = sapply(email_split, function(x) x[[2]])
domain_names
```

```
[1] "timshel.com"     "flinstones.com"
```

Typical things you might do with these operations

A lot of these operations can be done in text editors!

- Split a comma-delimited string into a list

- Parse HTML

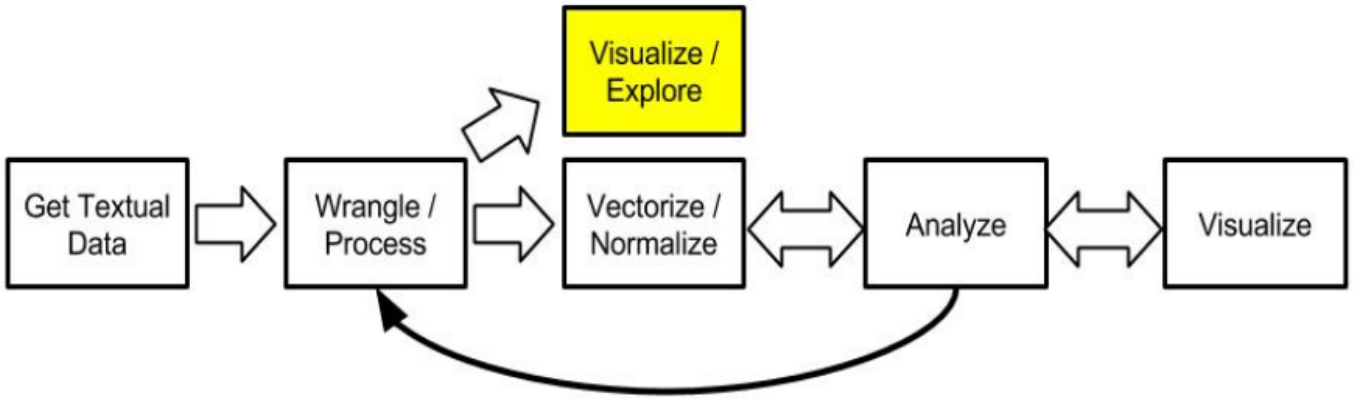
- Normalize your text by removing punctuation and all text lowercase

- Remove newlines or carriage returns and extra space

Why learn all of this manual text-processing?

- **Forms of the basis of text processing we'll see later**
- **Gives you the tools to customize!**
- **Can quickly get to word frequencies, which can be VERY informative**

Exploring + Visualizing Word Frequencies



Word Frequency Example

- Let's look at frequencies from 'Alice in Wonderland'
- Data comes from Project Gutenberg

Load and Pre-Process

```
alice <- "./pres_data/alice.txt"
#read into a character vector
alice <- paste(readLines(alice), collapse = "\n")
#Simple pre-processing
pre_process <- function(txt) {
  txt <- tolower(txt)
  txt <- str_replace_all(txt, "\n", " ") #newline with space
  txt <- str_replace_all(txt,"[ \t]{2,}", " ") #extra spaces
  txt <- str_replace_all(txt, "[^a-zA-Z ]","") #only keep letters
  return(txt)
}
#Pre-process
alice <- pre_process(alice)
```

Word Frequency Continue

Get frequencies

```
words <- str_split(alice," ")
freqs <- table(words)
freqs <- freqs[order(freqs,decreasing = T)]
freqs[1:8]
```

```
words
 the  and   to    a  she   it   of said
1631  844  721  627  537  526  508  462
```

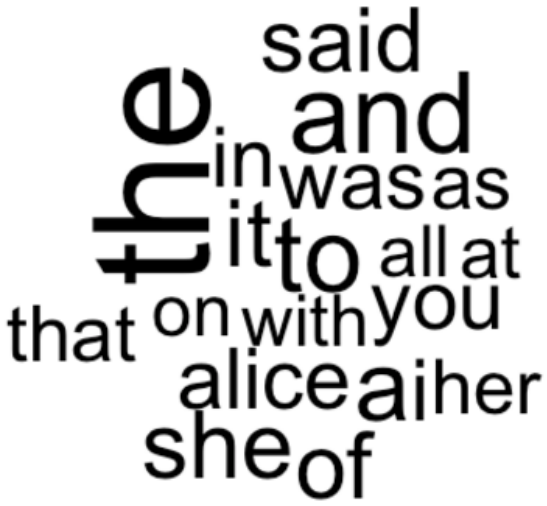
What do you notice?

Plotting word frequencies with wordclouds

Using the **`wordcloud`** package

We *COULD* use our frequency table…

```
library(wordcloud)
wordcloud(words = names(freqs), freq = as.numeric(freqs), max.words =
20, scale = c(8,3))
```



Plotting word frequencies with wordclouds

Or, we can just take advantage of the package :)

```
wc2 = wordcloud(alice, max.words = 20, scale = c(8,3))
```

Notice any differences?

Wordclouds with color!

Use two visual dimensions to emphasize frequency differences

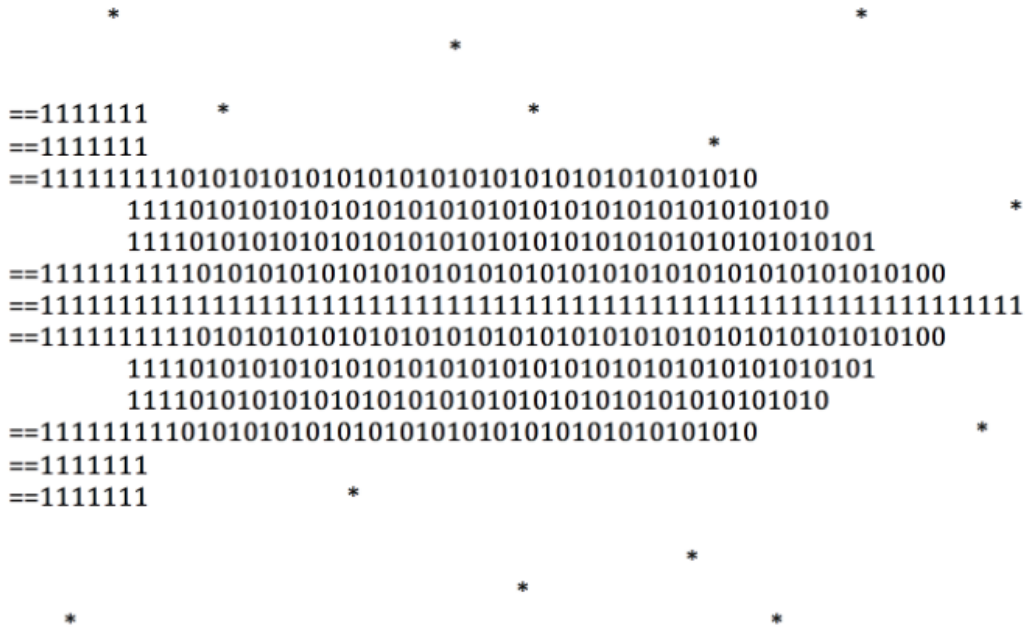
```
library(RColorBrewer)
pal <- brewer.pal(8,"Dark2")

wordcloud(alice, max.words = 20, scale = c(8,3), color = pal)
```

Why all this focus on word frequencies?

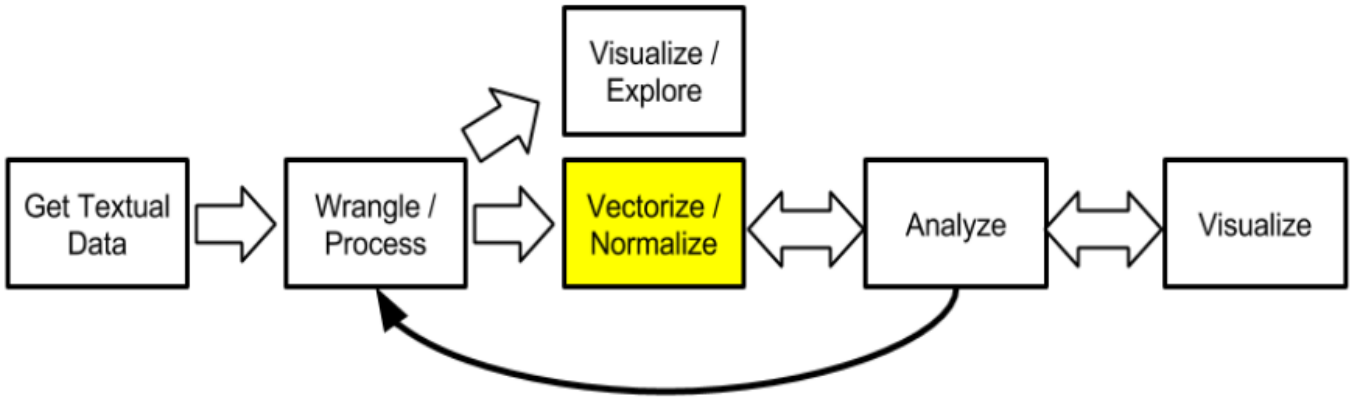
1. They're informative!

2. They are the basis of:

THE VECTOR-SPACE MODEL



Vectorizing Text



Terminology Reminder

- **Text**: the writen form of language
- **Document**: a bunch of text that comes from the same place
  - Can be as large as a whole book, or as small as a single tweet
- **Corpus**: A collection of documents (plural: *corpora*)

The Vector Space / N-Gram Model

Representing documents as vectors

```
doc 1: The cow jumped over the moon.
doc 2: The cows ate grass.
```

The document feature or document term matrix

```
Document-feature matrix of: 2 documents, 9 features.
2 x 9 sparse Matrix of class "dfmSparse"
       features
docs    ate cow cows grass jumped moon. over the The
  doc1    0   1    0     0      1     1    1   1   1
  doc2    1   0    1     1      0     0    0   0   1
```

What sorts of pre-processing do you think we need?

```
Document-feature matrix of: 2 documents, 9 features.
2 x 9 sparse Matrix of class "dfmSparse"
       features
docs    ate cow cows grass jumped moon. over the The
  doc1    0   1    0     0      1     1    1   1   1
  doc2    1   0    1     1      0     0    0   0   1
```

What sorts of pre-processing do you think we need?

It depends on what you're doing. But generally:

1. Remove punctuation
2. Lowercase
3. Tokenization
4. Remove stop-words (e.g., "a", "the", etc.)
5. Stemming
6. Remove low-frequency words

Tokenization

How you break up your text into meaningful units for analysis

Could be individual words

- Unigram / bag of words representation

Could be pairs or triples of words

- bigram / trigram

Could be sentences, paragraphs, etc.

Word stemming

Treat similar forms of a word:

run, runs, running, runner?

As the same word:

run

Reasons for doing it:

- Reduce vocabulary size
- Increase similarity across texts

Reasons to not do it:

- Lose information

Word stemming - main types

Remove suffixes from words

run, *runs*, *running*, *runner* -> run

Common algorithms: Porter, Snowball, Lancaster, regexp

Lemmatization

Use syntactic analysis to remove inflectional (i.e., sytnactic) endings

am, is, are -> be

Vectorizing documents

- Thankfully, all of this pre-processing is built into packages / libraries that handle text!

- Today we'll use the `quanteda` package

```
library(quanteda)
txt  = c("The cow jumped over the moon.", "The cows ate grass")
#Convert text list to a corpus
txt_corpus = corpus(txt, docnames = c("doc1","doc2"))
doc_mat2 = dfm(txt_corpus, clean = T, stem = T, ignoredFeatures =
stopwords("english"), verbose = F)
doc_mat2
```

```
Document-feature matrix of: 2 documents, 5 features.
2 x 5 sparse Matrix of class "dfmSparse"
       features
docs    ate cow grass jump moon
  doc1    0   1     0    1    1
  doc2    1   1     1    0    0
```

Effects of pre-processing

No Pre-Processing

```
Document-feature matrix of: 2 documents, 9 features.
2 x 9 sparse Matrix of class "dfmSparse"
       features
docs    ate cow cows grass jumped moon. over the The
  doc1    0   1    0     0      1     1    1   1   1
  doc2    1   0    1     1      0     0    0   0   1
```

With Pre-Processing

```
Document-feature matrix of: 2 documents, 5 features.
2 x 5 sparse Matrix of class "dfmSparse"
       features
docs    ate cow grass jump moon
  doc1    0   1     0    1    1
  doc2    1   1     1    0    0
```

No need to restrict ourselves to single words

- **So far what we've done is a unigram, or bag-of-words model**
- **Can also incorporate bigrams**

```
Document-feature matrix of: 2 documents, 8 features.
2 x 8 sparse Matrix of class "dfmSparse"
       features
docs    ate ate_grass cow cow_jump grass jump jumped_ov moon
  text1   0         0   1        1     0    1         1    1
  text2   1         1   1        0     1    0         0    0
```

If you want to move to trigrams, check out the `tm` package

Food for thought: ***What happens to our matrix as we add bigrams, trigrams, etc?***

What happens if documents are different lengths?

A corpus of Alice in Wonderland and our single sentence used before

```
library(quanteda)
txt  = c("The cow jumped over the moon.", alice)
txt_corpus = corpus(txt)
doc_mat3 = dfm(txt_corpus, clean = T, stem = F, ignoredFeatures =
stopwords("english"), verbose = F)
sort(doc_mat3)[,1:10]
```

```
Document-feature matrix of: 2 documents, 10 features.
2 x 10 sparse Matrix of class "dfmSparse"
       features
docs    said alice little one know like went thought queen time
  text1    0     0      0   0    0    0    0       0     0    0
  text2  462   385    128 101   85   85   83      74    68   68
```

Normalizing frequencies

- **Normalize by frequency within each document (i.e., row of our document-term matrix)**

- **There are many ways you could do this**

Relative frequency

```
relFreq = weight(doc_mat3, type = "relFreq")
sort(relFreq)[,1:5]
```

```
Document-feature matrix of: 2 documents, 5 features.
2 x 5 sparse Matrix of class "dfmSparse"
       features
docs           jumped         moon       cow       said      alice
  text1 0.3333333333 3.333333e-01 0.3333333 0          0
  text2 0.0004747962 7.913271e-05 0         0.03655931 0.03046609
```

Normalizing frequencies

Maximum frequency

```
relFreq = weight(doc_mat3, type = "relMaxFreq")
sort(relFreq)[,1:5]
```

```
Document-feature matrix of: 2 documents, 5 features.
2 x 5 sparse Matrix of class "dfmSparse"
       features
docs         jumped        moon cow said     alice
  text1 1.00000000 1.000000000   1    0 0
  text2 0.01298701 0.002164502   0    1 0.8333333
```

Other common options:

- natural log
- length normalization

Term Frequency Inverse Document Frequency (TFIDF)

A go-to approach for many NLP tasks

1. Normalize by frequency within each document (i.e., by row)
2. Normalize by *inverse* frequency across documents (i.e., by column)
   - `-log(1 / (# of docs term appears in))`

Term Frequency Inverse Document Frequency (TFIDF)

The effect of TFIDF is to emphasize words that frequent within a document, and infrequent across documents

Term Frequencies

	is	dog	lion	egg	Max Frequency
doc1	30	10	10	0	30
doc2	20	5	40	0	40
doc3	10	0	0	30	30
document frequency	3	2	2	1	
IDF	0	0.176	0.176	0.477	

TFIDF

	is	dog	lion	egg
doc1	0	0.059	0.059	0
doc2	0	0.022	0.176	0
doc3	0	0	0	0.477

Note: Term frequencies above were normed by max frequency

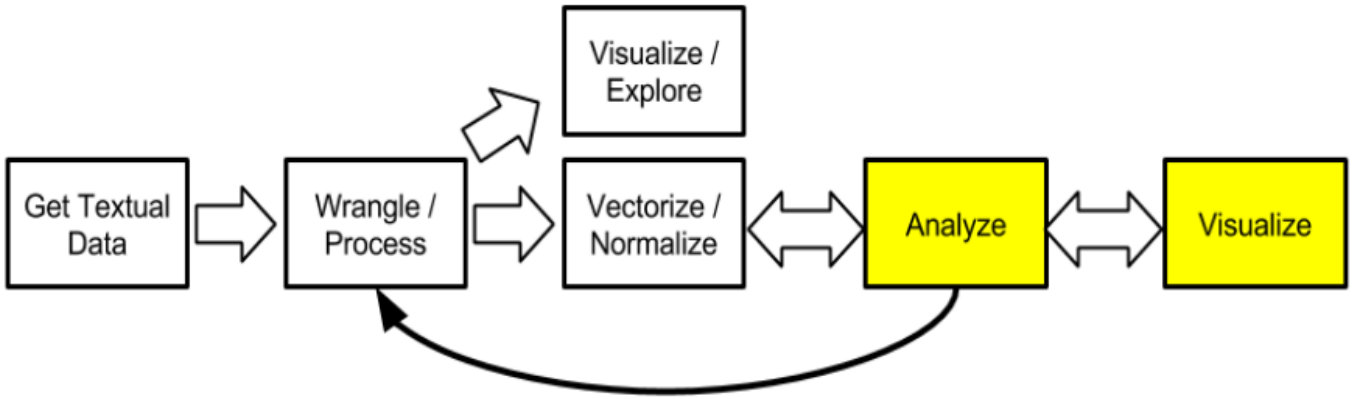
Term Frequency Inverse Document Frequency (TFIDF)

```
tfidf_norm = weight(doc_mat2, type = "tfidf")
sort(tfidf_norm)[,1:5]
```

```
       features
docs           ate     grass      jump      moon cow
  text1 0.0000000 0.0000000 0.2310491 0.2310491   0
  text2 0.2310491 0.2310491 0.0000000 0.0000000   0
attr(,"weighting")
[1] "tfidf"
attr(,"class")
[1] "dfm"    "matrix"
```

With our text now vectorized...it's time for some fun!

Analyzing Text with Machine Learning



Machine Learning with Text

After all the vectorizing, we end up with a matrix of features to use for many different tasks:

- **Unsupervised learning:**
  - clustering
  - document similarity
  - semantic analysis
- **Supervised learning:**
  - sentiment analysis
  - document classification / tagging
- **Information retrieval**

Clustering / Document Similarity

Answers the question: Which of our documents group together based the language they contain?

Basic Procedure

1. Clean and vectorize text

(2. Create a distance / similarity matrix)

3. Cluster

Clustering Example: Tweets about sustainable development

The data:

Downloaded from the Twitter search API looking for *"sustainable development"* and *"SDG"*

```
tweets = read.csv("pres_data/twitter_dat.csv", stringsAsFactors = F)
print(tweets$text[1])
```

```
[1] "RT @CUBAONU: #CPD49 #Cuba proves that sustainable development
can be reached  w/ political will https://t.co/fatAie4Gyr
#UNPopulation"
```

```
dim(tweets)
```

```
[1] 20062     2
```

```
#Let's remove retweets
RTs <- sapply(tweets$text, function(x) str_detect(x, "RT "))
tweets <- tweets[!(RTs),]
dim(tweets)
```

```
[1] 8771    2
```

Clustering Example Step 1a: Clean

The clean tweet function below is a modification from this blogpost

```
clean_tweets <- function(txt) {
  #Args:
  #  txt: character vector of text from twitter
  #Returns:
  # txt:  cleaned character vector of text

  # remove retweet entities
  txt = str_replace_all(txt, "(RT|via)((?:\\b\\W*@\\w+)+)", " ")
  # remove at people
  txt = str_replace_all(txt,"@\\w+", " ")
  # remove html links
  txt = str_replace_all(txt,"http\\S+", " ")
  # remove punctuation
  txt = str_replace_all(txt,"[[:punct:]]", " ") #or [^a-zA-Z0-9]+
  # remove numbers
  txt = str_replace_all(txt, "[[:digit:]]", " ") #or [0-9]+
  # remove unnecessary spaces
  txt = str_replace_all(txt,"[ \t]{2,}", " ")
  txt = str_trim(txt,"both")
  # remove single character words remaining
  txt = str_replace_all(txt, " [a-zA-Z] "," ")
  txt = tolower(txt)
  return(txt)
```

Clustering Example Step 1a: Clean

```
tweets$text_clean = sapply(tweets$text, function(x) clean_tweets(x))
print(tweets$text[1:2])
```

```
[1] "#CPD49 #Cuba proves that sustainable development can be reached
w/ political will https://t.co/fatAie4Gyr #UNPopulation"
[2] "What are the #globalgoals? The full list is here.
https://t.co/mXdcm574IN"
```

```
print(tweets$text_clean[1:2])
```

```
[1] "cpd cuba proves that sustainable development can be reached
political will unpopulation"
[2] "what are the globalgoals the full list is here"
```

Clustering Example Step 1b: Vectorize

Choice of how you vectorize will depend on the simililarity metric you use

- **frequency** or **tfidf** coded
- **binary** coding

Some choices during pre-processing:

- stem?
- removed words
- frequency cutoffs

Clustering Example Step 1b: Vectorize

```
tweet_corpus = corpus(tweets$text_clean)
tweet_dfm = dfm(tweet_corpus, clean = T, verbose = F,
                ignoredFeatures =
c(stopwords("english"),"sustainable","development","goals"))
dim(tweet_dfm)
```

```
[1] 20062 10913
```

```
#remove terms with low frequency across documents
tweet_dfm = trim(tweet_dfm, minDoc = 40, verbose = T)
```

```
Features occurring in fewer than 40 documents: 10197
```

```
tweet_dfm = weight(tweet_dfm, method = "tfidf")
```

Clustering Example Step 2: Calculate distance / similarity

- Many different ways of calculating distance / similarity:

- Euclidean
- Jaccard
- Cosine

- Distance calculations can take a while depending on the size of the matrix

- Reduce the size of the matrix
- Consider trying to parallelize

Clustering Example Step 2: Calculate distance / similarity

- The proxy package has a number of distance / similarity measures available

- Here we use Euclidean distance after scaling each column

- this is roughly equivalent to cosine similarity

```
library(proxy)
## Only going to work with a subset of 3000 tweets
tweet_dfm_sub = tweet_dfm[1:3000, ]
sim_mat = dist(scale(tweet_dfm_sub))
```

Clustering Example Step 3: Hierarchical Clustering

- Here we'll use hierarchical clustering on our distance matrix

- Challenges:

- where to make the cuts / how many clusters?
- how to visualize?

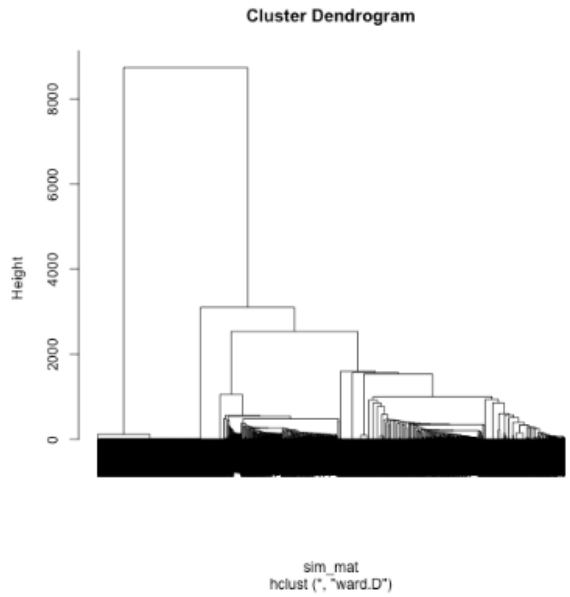
Perform hierarchical clustering

```
hier_clust = hclust(sim_mat, method="ward.D")
```

Visualizing Hierarchical Clusters: Dendrograms

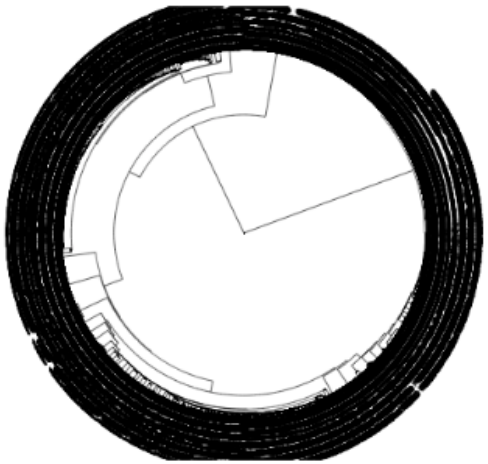
From base R `hclust`

```
plot(hier_clust, labels = F)
```



Fan plot from ape package

```
library(ape)
plot(as.phylo(hier_clust), type
= "fan")
```



Visualizing Clusters of Text with Wordclouds

Steps:

1. Assign documents to a group by cutting the tree
2. Look at wordclouds for each group

```
# Cut into 20 groups
groups = cutree(hier_clust, k =
20)

# Plot wordclouds
par(mfrow = c(3, 3))
for (group in seq(1, 17, 2)) {
    group_dat =
tweet_dfm_sub[which(groups ==
group)]
    plot(group_dat, max.words =
20)
}
```

Clustering Example Step 3: KMeans

- One advantage of using KMeans: No similarity matrix needed!

- Still need to think about:

- preprocessing
- Choice of K
- Visualizing

```
k = 20
kmeans_clust = kmeans(tweet_dfm[1:3000,], centers = k)
groups = kmeans_clust$cluster
```

Clustering Example 3: KMeans Plotting

Semantic Analysis through topic modeling

Answers the question: What is the underlying meaning captured in different documents?

Topic modeling is a statistical approach to natural language processing that find patterns in documents where words within the document can be assigned to abstract 'topics'

A document about sports is likely to have a different distribution of words than a document about cooking

Approaches to topic modeling

Latent Semantic Indexing (LSI) / Latent Semantic Analysis (LSA)

- uses singular value decomposition (SVD) over a document-term matrix
- equivalent to multivariate principle components analysis (PCA)

Latent Dirichlet Allocation (LDA)

- a generative (i.e, probabilistic) framework for estimating topics
- assumes each document is a mixture of topics defined by the words in the document
- most commonly used today

Latent Dirichlet Allocation (LDA) - Algorithm

Initialization

- Choose the number of topics
- Randomly assign topics to words within a document according to a dirichlet distribution
- Gives you the distribution of words over topics and distribution of topics over documents

Learning - iteratively repeat the following:

For each document calculate:

- Probabality of each topic given the words in the document
- For each word, the probability of the different topics based on the topic assignment of that word in the other documents

Text pre-processing for LDA

- LDA works off of raw term frequencies. So, tfidf is out here.

- Preprocessing:

- lowercase, punctuation and stop-word removal
- tokenization
- stemming
- Removing low frequency words
  - document occurence
  - using tfidf cutoff

LDA over DoGoodData session descriptions

Let's have some fun with DoGoodData's data!

Data is a webscrape of the session description

This is a toy example, and is not the right data for topic modeling

- there aren't enough documents
- text is sometimes not very long

LDA over DoGoodData session descriptions

All that being said...today we'll use the `topicmodels` library

Create a document-term matrix as before

```
library(topicmodels)
dgd_txt = textfile("./pres_data/DoGoodData_sessions/*.txt", cache =
F)
dgd_corpus1 = corpus(dgd_txt)
dgd_dfm1 = dfm(dgd_corpus1, clean = T, verbose = F,
               ignoredFeatures = stopwords("english"))
paste0("Matrix dimensions: ", paste(dim(dgd_dfm1), collapse = " X "))
```

```
[1] "Matrix dimensions: 80 X 2645"
```

Run the topic model with 10 topics

```
dgd_LDA10 <- LDA(convert(dgd_dfm1, to = "topicmodels"), k = 10)
```

LDA1 - Results

View top 5 terms for each document

```
get_terms(dgd_LDA10,5)
```

```
     Topic 1    Topic 2          Topic 3    Topic 4     Topic 5
[1,] "data"     "data"           "will"     "data"      "data"
[2,] "will"     "will"           "data"     "will"      "will"
[3,] "can"      "organizations"  "open"     "solar"     "change"
[4,] "well"     "analysis"       "feedback" "community"
"organizations"
[5,] "learning" "can"            "sector"   "us"        "staff"
     Topic 6       Topic 7  Topic 8     Topic 9   Topic 10
[1,] "data"        "data"   "data"      "data"    "data"
[2,] "will"        "will"   "will"      "will"    "organizations"
[3,] "health"      "can"    "can"       "can"     "will"
[4,] "populations" "use"    "process"   "stories" "system"
[5,] "survey"      "impact" "analytics" "water"   "social"
```

What do you think?

"Improving" our LDA

1. Lots of repeated terms across topics

```
topfeatures(dgd_dfm1)
```

```
        data         will          can          use      session
         341          184           84           59           50
organizations        learn        using       social  information
          47           43           41           37           36
```

2. Definitely too many topics given our corpus size

"Improving" our LDA

```
top_remove =  names(topfeatures(dgd_dfm1)[1:8])
dgd_dfm2 = dfm(dgd_corpus, clean = T, verbose = F,
               ignoredFeatures = c(top_remove,stopwords("english")))
dgd_LDA4<- LDA(convert(dgd_dfm2, to = "topicmodels"), k = 4)
get_terms(dgd_LDA4,5)
```

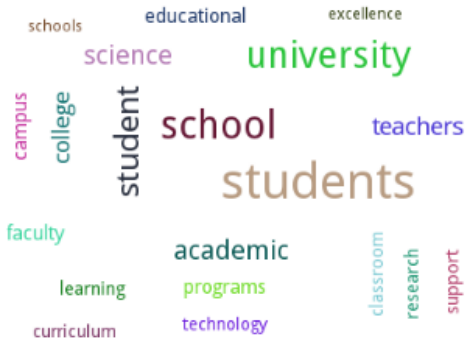
```
     Topic 1        Topic 2       Topic 3        Topic 4
[1,] "program"      "new"         "learning"     "nonprofit"
[2,] "programs"     "analysis"    "machine"      "tools"
[3,] "feedback"     "information" "organization" "solar"
[4,] "impact"       "change"      "well"         "community"
[5,] "participants" "crisis"      "work"         "open"
```

- **NOTE:** It might not be a good idea to remove terms as below. These represent real aspects of the topics in a corpus.
- It's done here to illustrate the differerences between topics

Visualizing Topic Models - wordclouds

Words are top X words for a topic.

Size correspond to the weight.



Visualizing Topic Models - networks

Nodes are words, edges are associations to the same topic

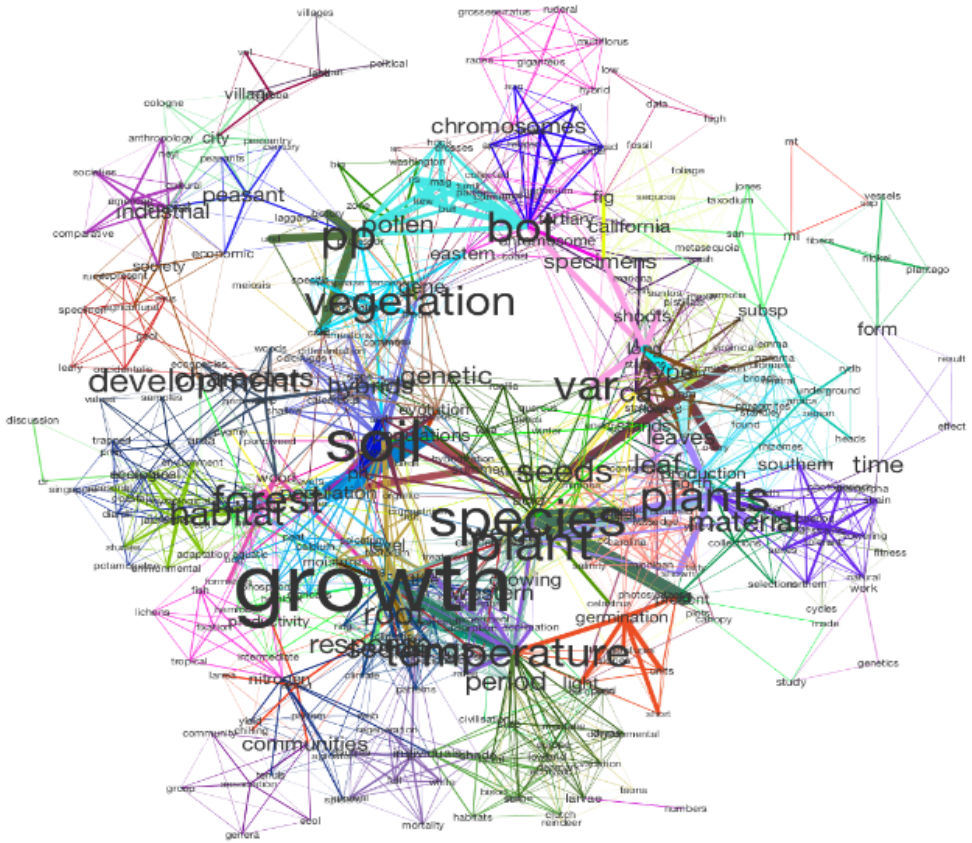


image comes from tetne tutorial

Visualizing Topic Models

- **LDAvis: a fantastic tool for both R and Python**
- **Trick is to get your topic model data in the right format**
- **Here, I modified some code from Christopher Gandrud (hidden below)**

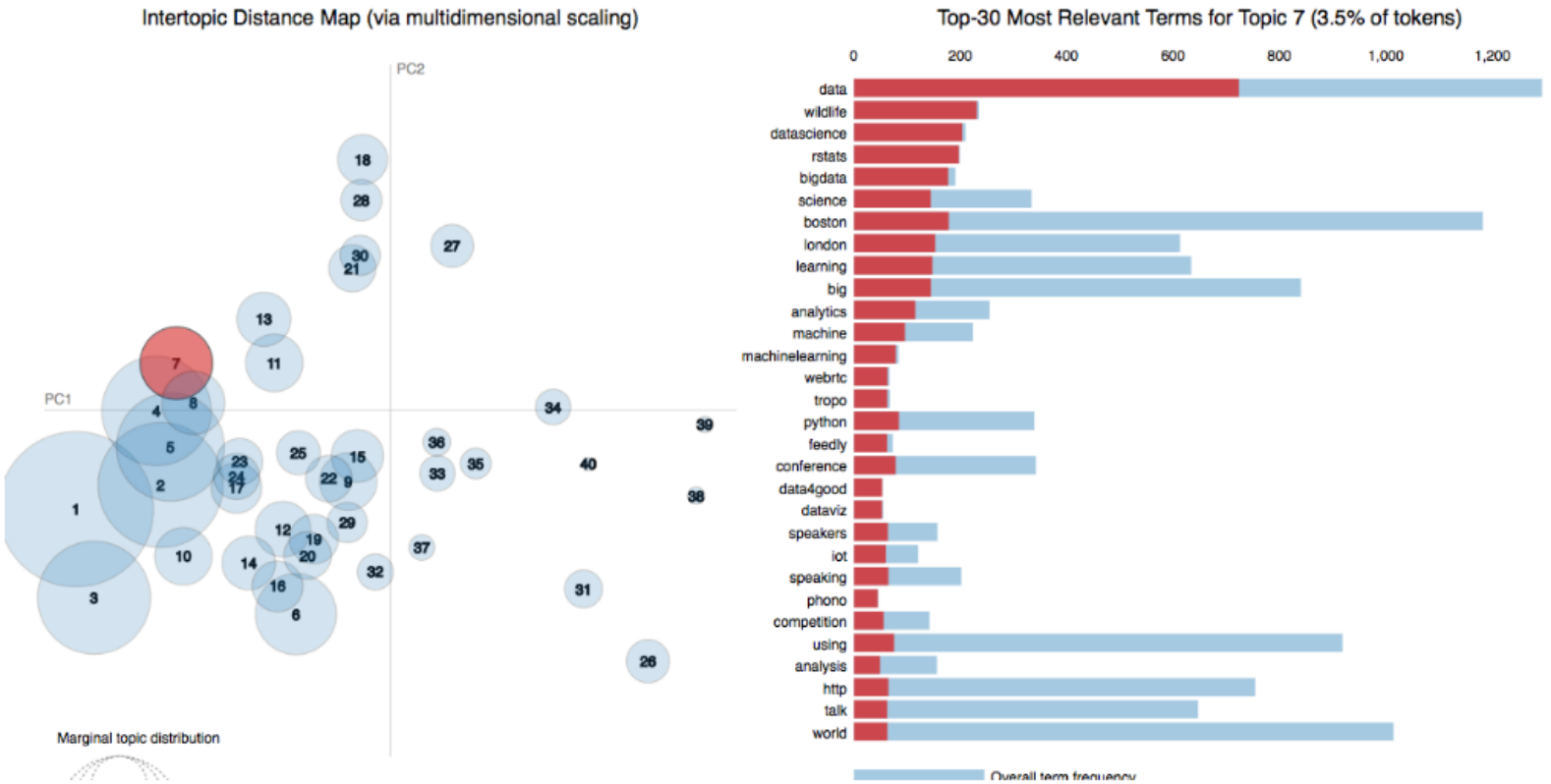
```
#Visualizing the original topic model
lda_vis_dat <- topicmodels_json_ldavis(fitted = dgd_LDA10, corpus =
dgd_corpus1, doc_term = dgd_dfm1)
```

```
Corpus consisting of 80 documents.
```

```
serVis(lda_vis_dat)
```

LDAvis demo

In browser…



How do you know if your model is right?

Hard to answer directly

1. Do the topics make sense?

2. *perplexity* on a hold-out set of documents

- how "confused" is your model?

3. Can the results be used for another task? E.g., supervised learning

Supervised learning with text

Using our vectorized text as features to learn about a known outcome variable

Today:

- Sentiment Analysis
- Document Classification

Supervised learning with text

Commonly-used algorithms:

- Naive Bayes
- Multinomial logistic regression (a.k.a. maximum entropy; maxent)
- Support Vector Machines

Sentiment Analysis

Answers the question: What is the emotional valence of a particular piece of text?

Old school:

- create a dictionary of positive / negative terms
- count frequency for each text source
- majority count wins or difference/ratio of positive to negative

Sentiment Analysis

Answers the question: What is the emotional valence of a particular piece of text?

- Positive / negative valence can be seen as an outcome variable

- This simply amounts to a supervised learning problem:

- **Categorization:** Sentiment classified as positive, negative or neutral
- **Regression:** Sentiment varies continuously from negative to positive

- Categorization is more frequent

Today: Classifying sentiments of Tweets

Data comes from the twitter sentiment corpus

Each tweet is categorized as negative ("0") or positive ("1")

```
library(data.table)
twit_dat = fread("./pres_data/Twitter_Sentiment_Data.csv")
dim(twit_dat)
```

```
[1] 50000     4
```

```
names(twit_dat)
```

```
[1] "ItemID"            "Sentiment"         "SentimentSource"
"SentimentText"
```

Today: Classifying sentiments of Tweets

The tweets:

```
twit_dat$SentimentText[1:5]
```

```
[1] "#niley wooooooo new untouchable #niley its really awesome #niley
it doesn't have any #niley in it though"
[2] "g'bye FL crew! until dragon*con!"
[3] "@apostropheme i'm a real BOY goddamit!!!!!!!!!!!!! guh.
apostro. i feel sad. the library lady thinks i'm stupid. SHE'S
STUPID.  j"
[4] "&amp;quot;PembsDave He's gone now"
[5] "@Abigailjune92 Weeeeeellllll hellllllllo abbiie! No one ever
tells me they have twitter and I've had it for ages. Hope you're not
too ill"
```

Data preprocessing

```
twit_dat$text_clean <- clean_tweets(twit_dat$SentimentText)

tweet_corpus = corpus(twit_dat$text_clean)
tweet_dfm = dfm(tweet_corpus, clean = T, verbose = F,
                ignoredFeatures = c(stopwords("english")))
dim(tweet_dfm)
```

```
[1] 50000 36206
```

```
#remove terms with low frequency across documents
tweet_dfm = trim(tweet_dfm, minDoc = 200, verbose = T)
```

```
Features occurring in fewer than 200 documents: 35921
```

```
tweet_dfm = weight(tweet_dfm, method = "tfidf")
```

Train a Naive Bayes classifier

Using the `e1071` library

```
library(e1071)

#Split into train and test
train_x = tweet_dfm[1:40000]
test_x = tweet_dfm[40001:50000]
train_y = twit_dat$Sentiment[1:40000]
test_y = twit_dat$Sentiment[40001:50000]

nb = naiveBayes(x = as.matrix(train_x), y = as.factor(train_y))
preds = predict(nb, newdata = as.matrix(test_x), type = 'class')
```

Model Performance

Accuracy

```
accuracy = mean(preds == as.factor(test_y))
print(paste0("Accuracy: ", accuracy))
```

```
[1] "Accuracy: 0.6659"
```

Confusion Matrix

```
conf_mat = table(preds, as.factor(test_y), dnn= c("pred","actual"))
print("Confusion Matrix:")
```

```
[1] "Confusion Matrix:"
```

```
conf_mat
```

```
    actual
pred    0    1
   0 2946 1265
   1 2076 3713
```

How could we improve our model?

Think about the *features* being used here

- “@apostropheme i'm a real BOY goddamit!!!!!!!!!!!!!! guh. apostro. i feel sad. the library lady thinks i'm stupid. SHE'S STUPID. j”
- “@Abigailjune92 Weeeeeellllllll hellllllllo abbiie! No one ever tells me they have twitter and I've had it for ages. Hope you're not too ill”
- “awwwwwwwwwwwww that is so beautiful. I just need to be in his arms tonight”

Let's modify our data cleaning and preprocessing a bit

Keep punctuation

```
txt = "i'm a real BOY goddamit!!!!!!!!!!!!!!"
txt = str_replace_all(txt,"([:punct:])"," \\1 ")
print(txt)
```

```
[1] "i ' m a real BOY goddamit !  !  !  !  !  !  !  !  !  !  !  !  !
! "
```

Normalize repeated letters

```
txt = "awwwwwwwwwwww that is so beautiful."
txt = str_replace_all(txt,"([a-z])\\1+","\\1\\1")
print(txt)
```

```
[1] "aww that is so beautiful."
```

Keep stopwords

- they include 'not'

Stem

Train a Naive Bayes classifier with new features

Pre-Processing as described above

```
twit_dat$text_clean <- clean_tweets2(twit_dat$SentimentText)

tweet_corpus = corpus(twit_dat$text_clean)
tweet_dfm = dfm(tweet_corpus, clean = T, verbose = F, stem = T)
#remove terms with low frequency across documents
tweet_dfm = trim(tweet_dfm, minDoc = 200, verbose = T)
```

```
Features occurring in fewer than 200 documents: 28742
```

```
tweet_dfm = weight(tweet_dfm, method = "tfidf")
```

Train model as before (behind the scenes)

Model 2 Performance

Accuracy

```
accuracy2 = mean(preds == as.factor(test_y))
cat(paste0("Accuracy1: ", accuracy, "\nAccuracy2: ", accuracy2))
```

```
Accuracy1: 0.6659
Accuracy2: 0.681
```

Confusion Matrix

```
conf_mat2 = table(preds, as.factor(test_y), dnn= c("pred","actual"))
conf_mat2
```

```
    actual
pred    0    1
   0 3025 1193
   1 1997 3785
```

Well, that's a pretty modest improvement over the first model, but, sentiment is hard!

Some food for thought re: sentiment analysis

- As with any analysis: *Garbage in, Garbage out*

- some of our tweets aren't very good, nor is the scoring

- We're high-dimensional here: *More data = better*

- Lot's of nuance to sentiment that's hard to learn, e.g., sarcasm

Document classification

Answers the question: What category should this document be assigned to?

Examples:

- Categorize emails according to a known system
- Tag comment section / tweets / websites according to some taxonomy

Document classification

Answers the question: What category should this document be assigned to?

- From a machine learning standpoint, this is the same thing as sentiment analysis using a categorical outcome
- Often multi-category / multi-nomial
- Otherwise, same principles apply!

- From a machine learning standpoint, this is the same thing as sentiment analysis using a categorical outcome
- Often multi-category / multi-nomial
- Otherwise, same principles apply!

Summary

Today we covered:

1. Wrangling your text
   - Regular expressions are huge here
2. The Vector Space model
   - Lots of pre-processing
3. Machine Learning with text
   - Unsupervised clusering and LDA
   - Supervised learning with sentiment analysis

Moving forward

Explore other ways of vectorizing text:

(both easier in Python)

1. Hashing vectorizers (available in sklearn)
2. word2vec and doc2vec (available in gensim)

We've scratched the surface of NLP!

Syntactic analysis

- Part-of-speech (POS) tagging
- Syntactic parsing

Named entity recognition

- Treating people/places/proper nouns as a single entity

Machine translation

- E.g., Google translate

Speech recognition

- Combines all of the above and more

Natural language generation

- algorithms that can produce language

Thanks

Now go out and process some text!

To learn more about Timshel:

@timshel on Twitter

timshel.com

If you want to get in touch:

dan@timshel.com

@DataScience_Dan on Twitter