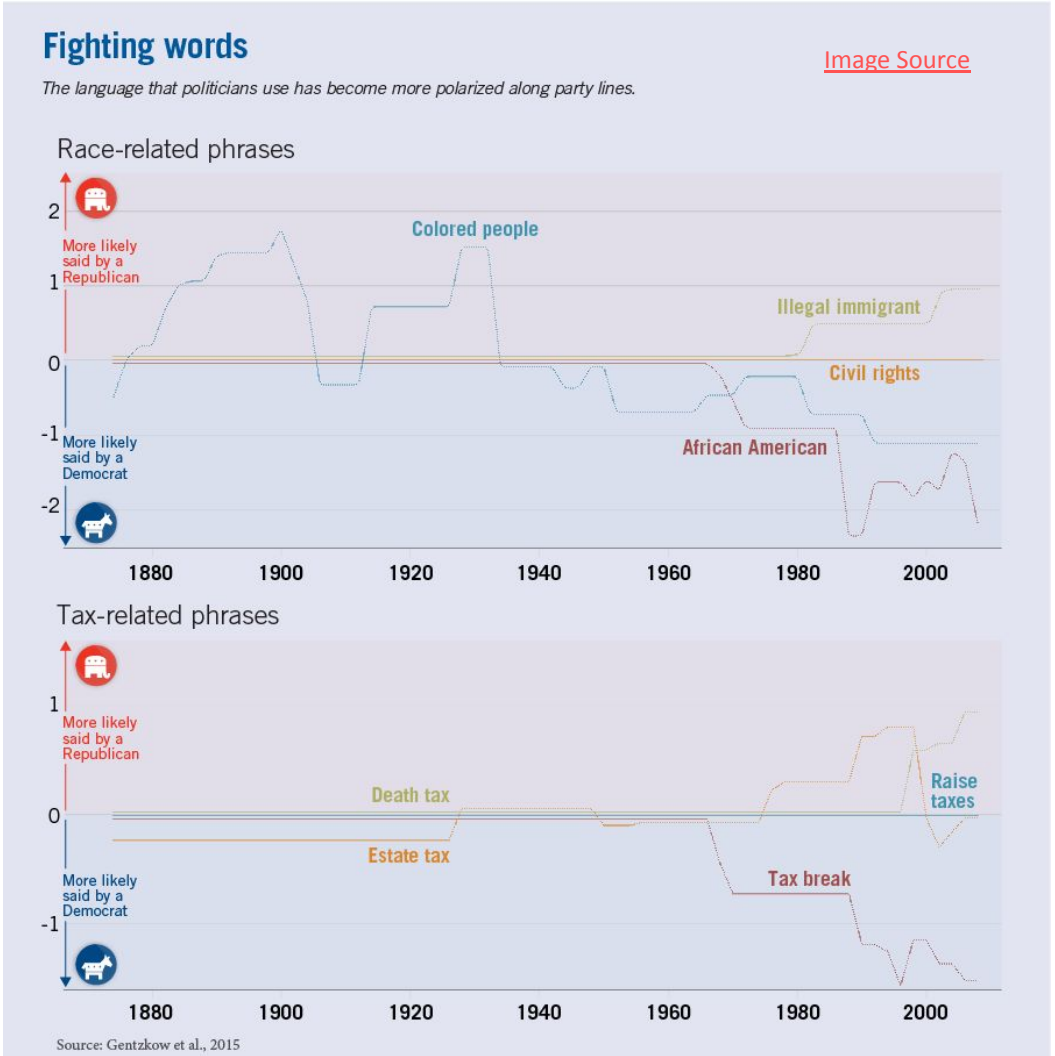
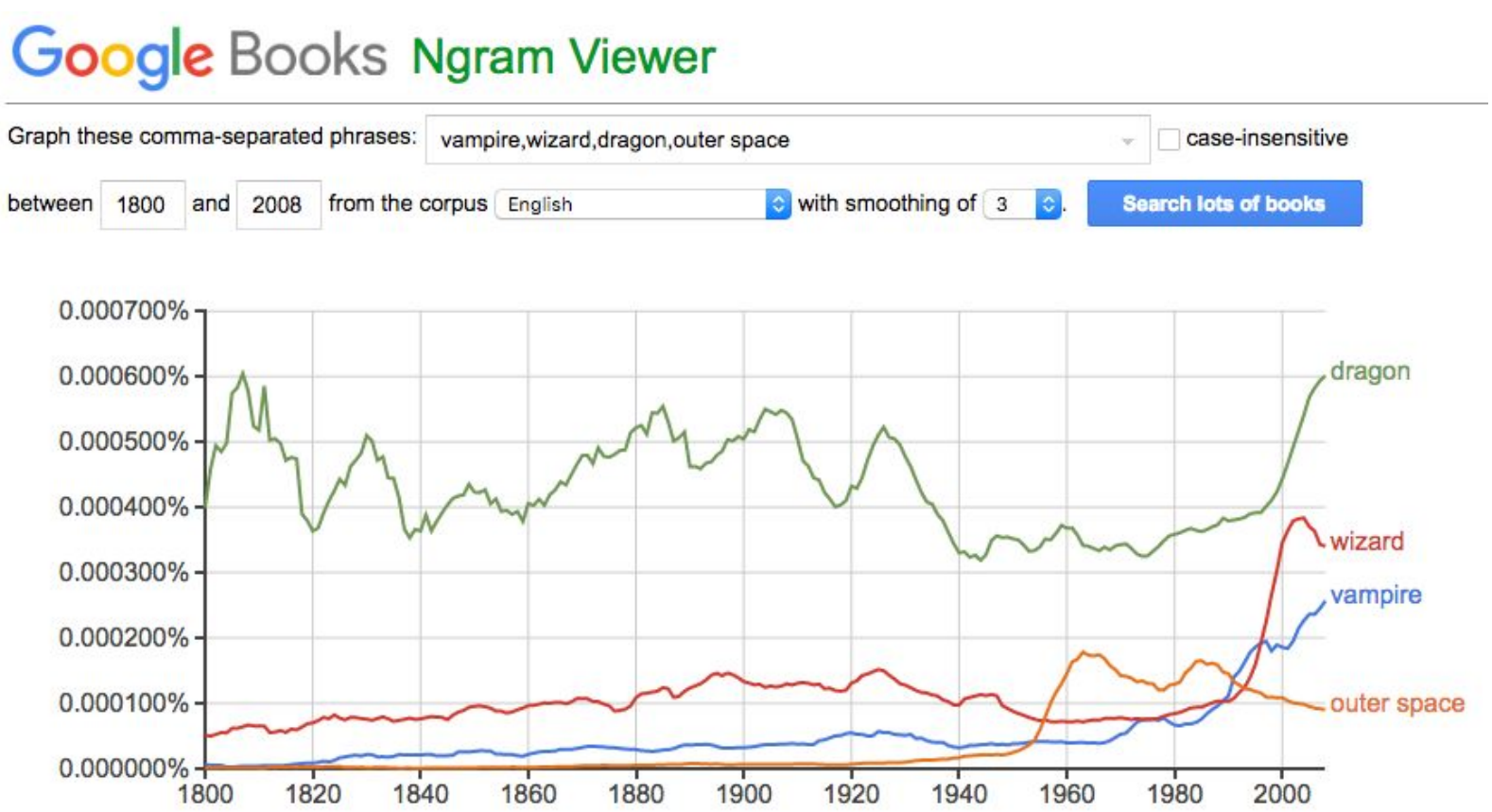
Text

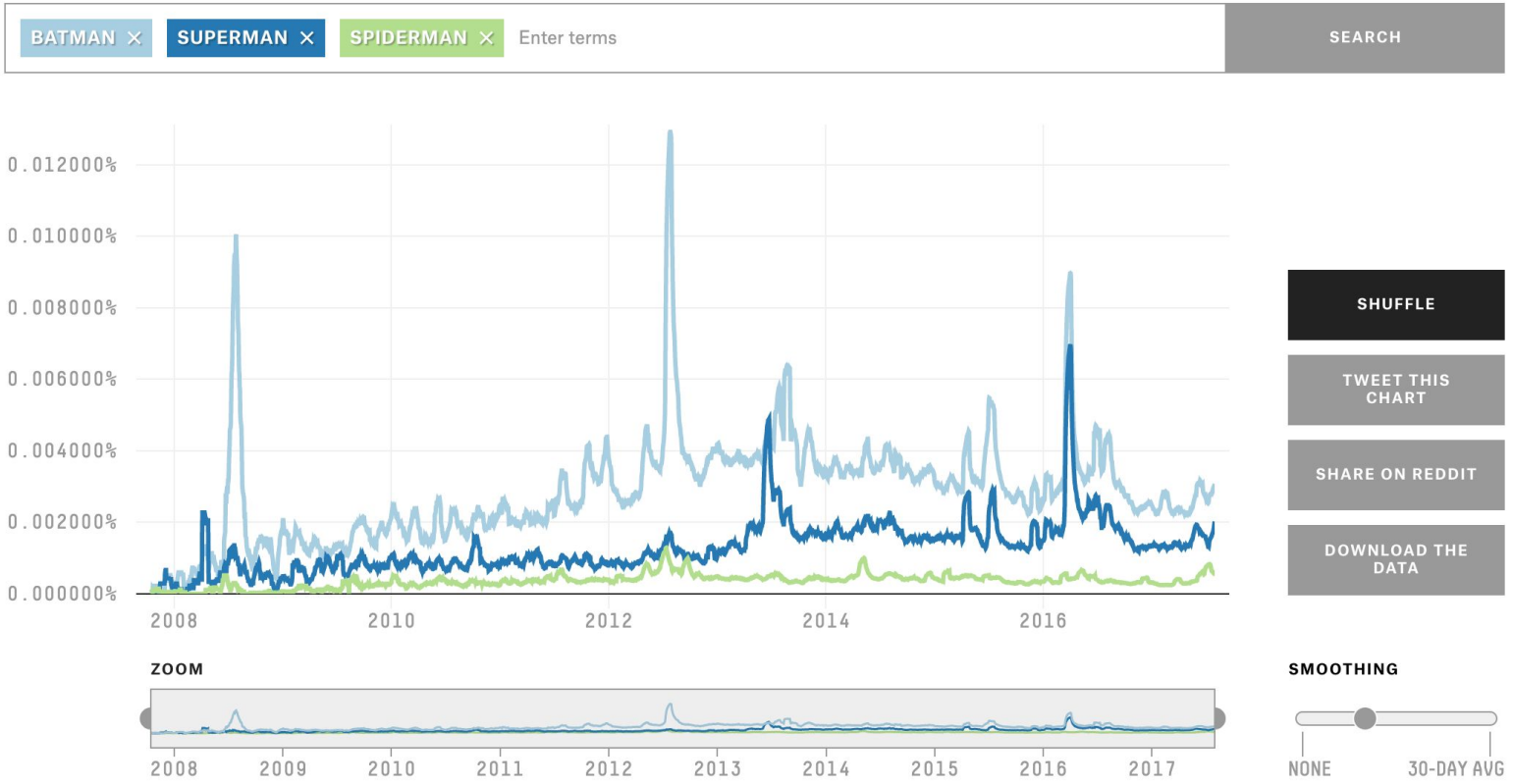
These lecture slides were compiled by Will Povell in 2017, when, as a sophomore, he TAed for CS 100.

How to talk like a Democrat (or a Republican)





Google Books Ngram Viewer: Charts word frequencies in books over time, offering insight into how language, culture, and literature have changed



How the Internet Talks: A Reddit Ngram viewer created by FiveThirtyEight to visualize the prevalence of terms on Reddit

Image Source

Power and Agency in Hollywood Characters

A team at UW analyzed the language in nearly 800 movie scripts, quantifying how much power and agency those scripts give to individual characters. Women consistently used more submissive language, with less agency.

"In the movie *Frozen*, only the princess Elsa is portrayed with high power and positive agency, according to a new analysis of gender bias in movies. Her sister, Anna, is portrayed with similarly low levels of power as 1950s-era Cinderella."

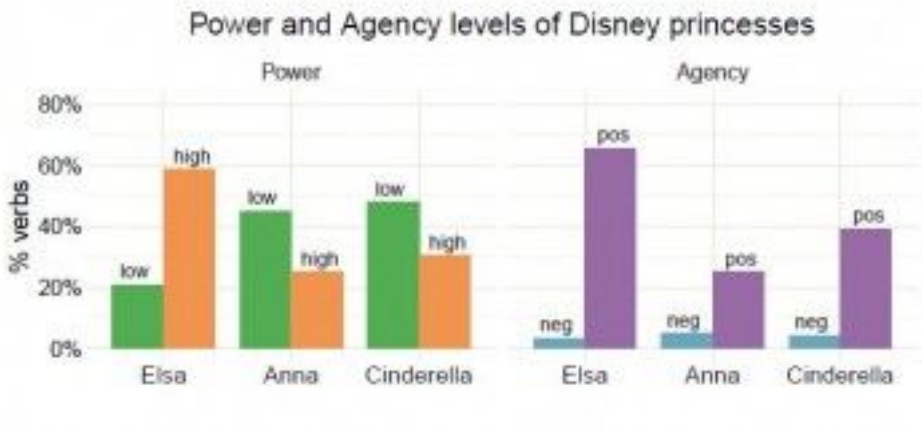
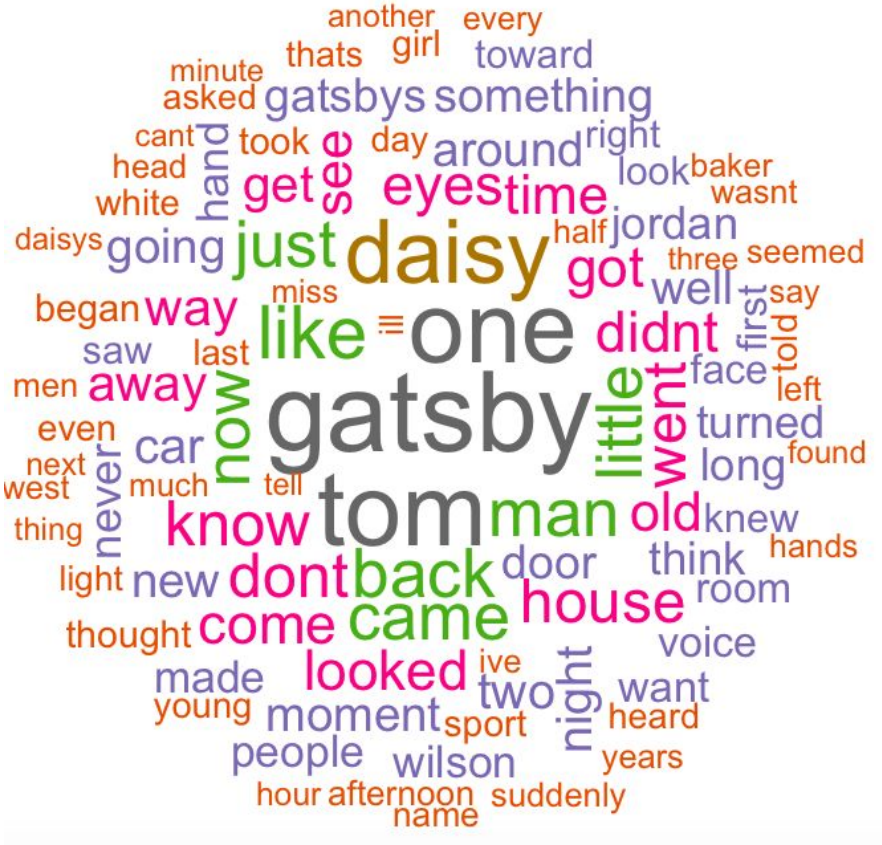


Image Source

Word Clouds

Visualizes words in a document with sizes proportional to how frequently the words are used

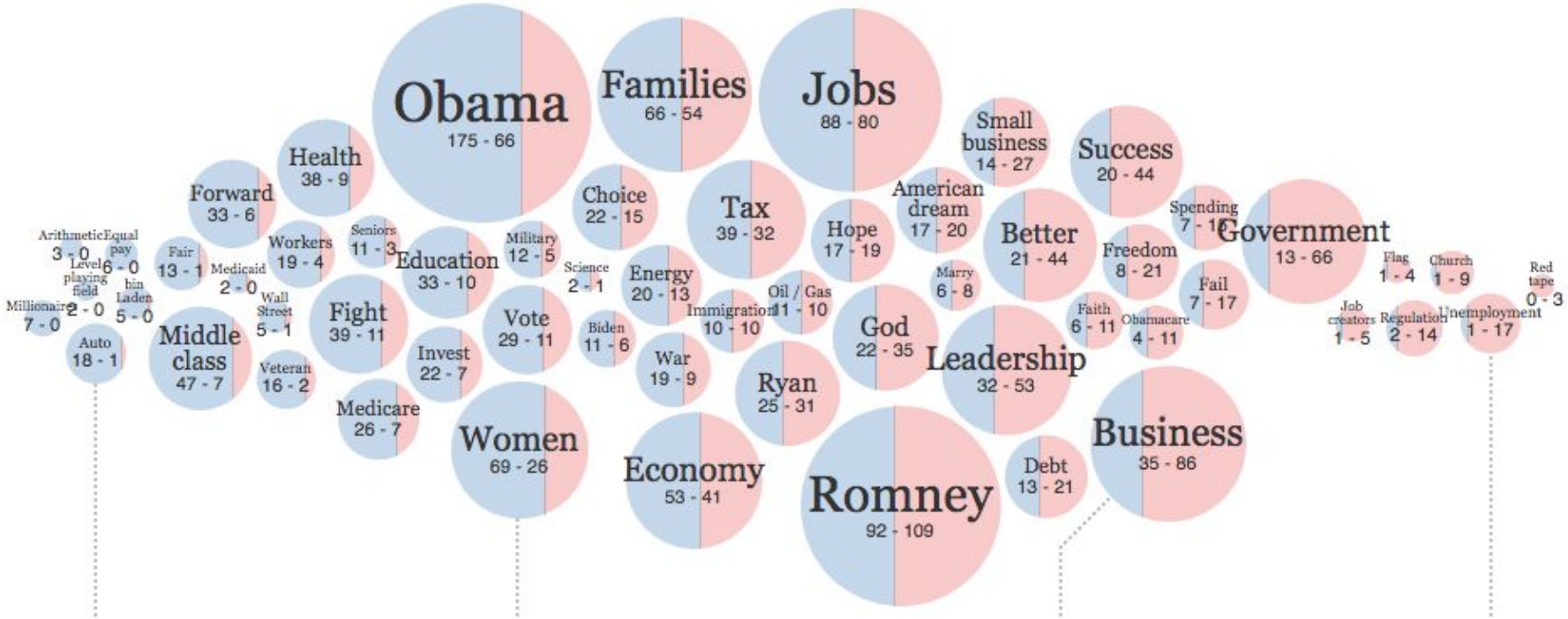
Example: *The Great Gatsby*



will
freedom
dream
ring
day
let
every
able
one
together
mountain
faith
shall
black
children
free
made
sing
god
today
nation
land
white
little
true
men
state
last
new
sons
boys
live
places
mississippi
sweltering
georgia
hands
heat
even
girls
alabama
hope
brotherhood
meaning
join
mountainside
words
hill
former
stone
thee
tomorrow
selfevident
pennsylvania
slopes
oppression
thank
hilltops
must
hamlet
injustice
character
still
jews
rise
lookout
south
curvaceous
beautiful
american
equal
new
york
struggle
discords
owners
great
molehill
despair
table
knowing
almighty
work
back
low
nullification
dripping
racists
truths
rough
difficulties
crooked
interposition
snowcapped
rockies
skin
village
pray
exalted
alleghenies
tennessee
jangling
gentiles
deeply
brothers
mighty
valley
jail
glory
colorado
lord
see
tis
lips
right
symphony
creed
fathers
sit
hold
allow
justice
sweet
flesh
color
negro
old
died
face
prodigious
pilgrim
vicious
red
slaves
governor
country
rooted
sisters
hampshire
mountains
straight
slave
heightening
speed
content
catholics
oasis
protestants
transform
pride
hills
america
judged
stand
revealed
though
city
four
created
california
plain
become
happens
liberty
spiritual
transformed

Image Source

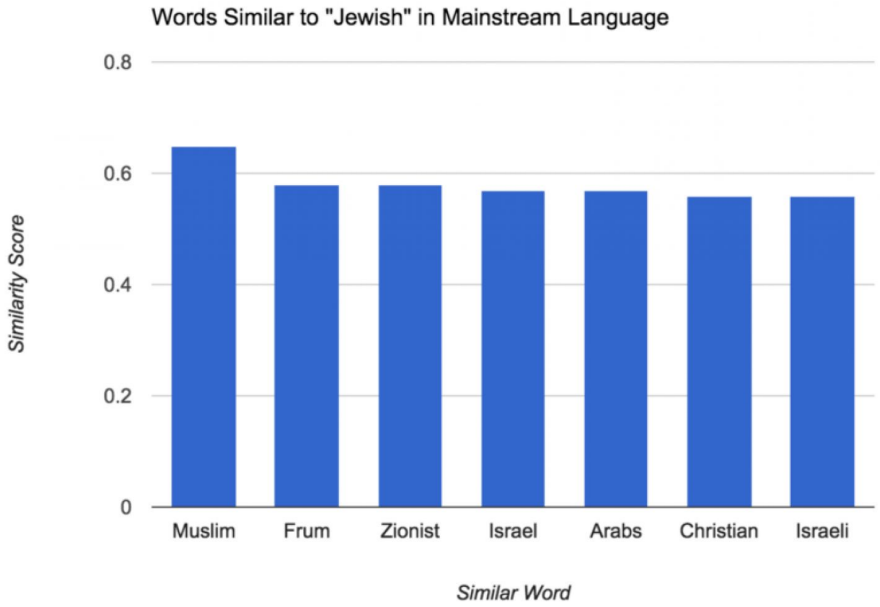
2012 Democratic and Republican Conventions



← Words favored by Democrats

Words favored by Republicans →

Language of the `alt-right`



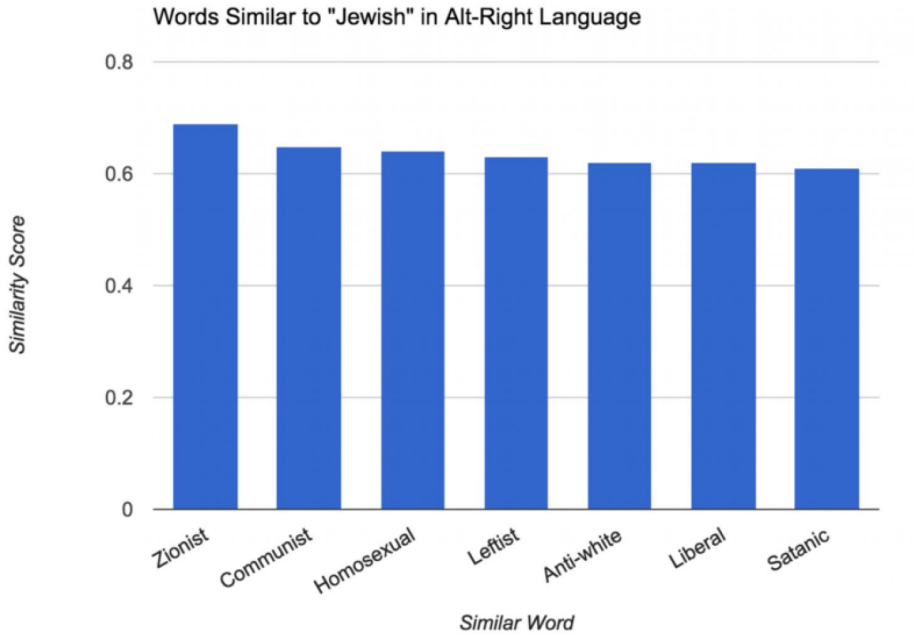


Image Source

Natural Language Processing

Natural Language Processing

Natural Language Understanding

- Analyze the syntactic structure of language and deriving semantic meaning
- Example tasks
  - Speech Recognition
  - Named Entity Recognition
  - Text Classification

Natural Language Generation

- Example tasks
  - Text Generation (a college essay written by PaLM or GPT)
  - Speech Generation (found in virtual assistants)
  - Question Answering

Text Preprocessing

Text Preprocessing (2017)

Jargon

- A set of documents is called a corpus (plural corpora).

The first step in text analysis is preprocessing (cleaning) the corpus:

- Tokenize: parse documents into smaller units, such as words or phrases
- Remove stop words (e.g., a, the, and, etc.) and punctuation
- Stemming & Lemmatization: standardize words with similar meaning
- Normalize: convert to lowercase (carefully: e.g., US vs. us)

Text Preprocessing (2022)

Jargon

- A set of documents is called a corpus (plural corpora).

The first step in text analysis is preprocessing (cleaning) the corpus:

- Lower casing
- Removal of Punctuations
- Removal of Stopwords
- Removal of Frequent words
- Removal of Rare words
- Stemming
- Lemmatization
- Removal of emojis
- Removal of emoticons
- Conversion of emoticons to words
- Conversion of emojis to words
- Removal of URLs
- Removal of HTML tags
- Chat words conversion
- Spelling correction

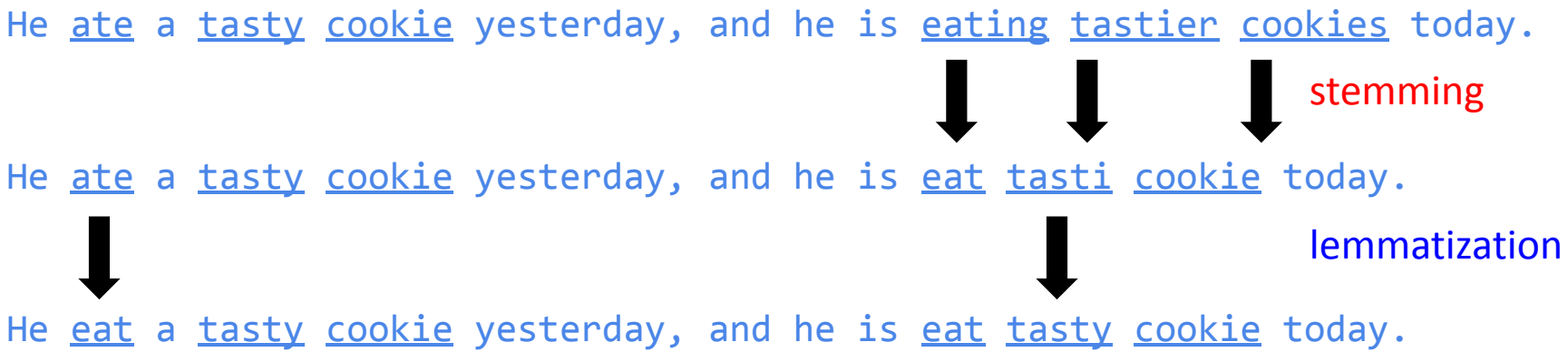
Noise Removal

Tokenization

- Split up a document into tokens
- Common tokens
  - Words: e.g., “hello”, “blueno”, “laptop”, etc.
  - Punctuation: e.g., . , “ ‘ ! ?, etc.
- Other tokens
  - Replace very uncommon words with an unknown token: <UNKNOWN>
  - End sentences (or sentence like structures) with a stop token: <STOP>
  - Replace all numbers with a single token: e.g., 100 → <NUM>
  - Replace common words (“a”, “the”, etc.) with <SWRD>
- Tokenization is a pain (there are lots of edge cases), but luckily, it is a solved problem
- “The dog ran in the park joyously!” → c(“<SWRD>”, “dog”, “ran”, “<SWRD>”, “<SWRD>”, “park”, “<UNKNOWN>”, “!”, “<STOP>”)

Stemming & Lemmatization

- Goal: standardize words with a similar meaning
- Stemming reduces words to their base, or root, form
- Lemmatization makes words grammatically comparable
  (e.g., am, are, is ⟹ be)



Normalization

Image Source

Raw	Normalized
2moro 2mrrw 2morrow 2mrw tomrw	tomorrow
b4	before
otw	on the way
:) :-) ;-)	smile

Examples:

- make all words lowercase
- remove any punctuation
- remove unwanted tags

```
Has Dr. Bob called? He is waiting for the results of Experiment #6.
                    ⬇
has dr bob called he is waiting for the results of experiment 6

<p>Text.</p><!-- Comment --><a href="#breakpoint">More text.</a>'
                    ⬇
text more text
```

A Final Note

Preprocessing should be customized based on the type of corpus.

- Tweets should be preprocessed differently than academic texts. Google Books Ngram Viewer vs. The Reddit Ngram Viewer
- So should the names of bands: e.g., The The.

Language Modeling

Bag-of-Words Model

Represents a corpus as an unordered set of word counts, ignoring stop words

Doc 1: David likes to play soccer.
Ben likes to play tennis.

Doc 2: Mary likes to ride her bike.

Doc 3: David likes to go to the movie theater.

	Doc 1	Doc 2	Doc 3
David	1	0	1
Mary	0	1	0
Ben	1	0	0
tennis	1	0	0
soccer	1	0	0
theater	0	0	1
bike	0	1	0
movie	0	0	1
ride	0	1	0
play	2	0	0
likes	2	1	1
go	0	0	1

N-gram Model

An *N*-gram is a sequence of *N* words in a corpus.

`The movie about the White House was not popular.`

N=1 (unigram, bag-of-words): The, movie, about, the, White, House, was, not, popular

N=2 (bigram): The movie, movie about, about the, the White, White House, House was, was not, not popular

N=3 (trigram): The movie about, movie about the, about the White, the White House, White House was, House was not, was not popular

N=4 …

TF-IDF: Term frequency-Inverse document frequency

Term frequency

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

- $f_{t,d}$: Frequency of term *t* in document *d*
- $\max\{f_{t',d} : t' \in d\}$: Maximum frequency of term *t* across all documents

Inverse document frequency

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

- N: Total number of documents
- $|\{d \in D : t \in d\}|$: Number of documents the term *t* appears in

tfidf(*t*, *d*, *D*) = tf(*t*, *d*) idf(*t*, *D*)

Useful for decreasing weight of common, low information words

TF-IDF: An Example

Document 1
The dog ate. The man walked.

Document 2
The dog ate happily. The dog ran through the park.

Document 3
The sun was shining. The birds were singing.

The

tfidf(“the”, 1, *D*) = tf(“the”, 1) idf(“the”, *D*)

tf(“the”, 1) = (2 / 3) ≈ 0.67
idf(“the”, *D*) = log(3 / 3) = 0

tfidf(“the”, 1, *D*) ≈ (0.67)(0) = 0

Dog

tfidf(“dog”, 1, *D*) = tf(“dog”, 1) idf(“dog”, *D*)

tf(“dog”, 1) = (1 / 2) ≈ 0.50
idf(“dog”, *D*) = log(3 / 2) ≈ 0.176

tfidf(“dog”, 1, *D*) ≈ (0.50)(0.176) ≈ 0.088

TF-IDF: An Example

Document 1
The dog ate. The man walked.

Document 2
The dog ate happily. The dog ran through the park.

Document 3
The sun was shining. The birds were singing.

Man

tfidf(“man”, 1, *D*) = tf(“man”, 1) idf(“man”, *D*)

tf(“man”, 1) = (1 / 1) = 1
idf(“man”, *D*) = log(3 / 1) ≈ 0.477

tfidf(“man”, 1, *D*) ≈ (1)(0.477) ≈ 0.477

Dog

tfidf(“dog”, 1, *D*) = tf(“dog”, 1) idf(“dog”, *D*)

tf(“dog”, 1) = (1 / 2) ≈ 0.50
idf(“dog”, *D*) = log(3 / 2) ≈ 0.176

tfidf(“dog”, 1, *D*) ≈ (0.50)(0.176) ≈ 0.088

Natural Language Understanding

Language is complex. The goal of text analysis is to strip away some of that complexity to extract meaning.

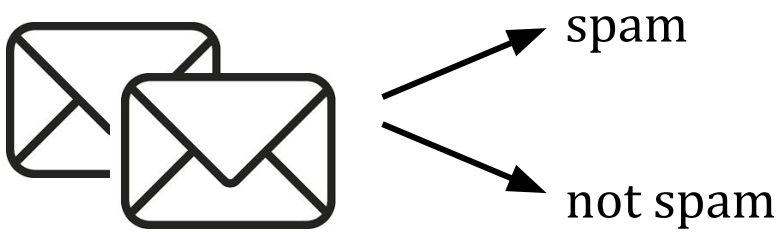
Text Analysis

The process of computationally retrieving information from text, such as books, articles, emails, speeches, and social media posts.

Document Classification

- Who wrote each of the Federalist papers (anonymous essays in support of the U.S. constitution)?
  - John Jay, James Madison, or Alexander Hamilton?
- A text analysis of the Federalist papers
  - An Analysis of word frequency, distribution, patterns, and meaning.

Spam Detection



Naive Bayes

- For all classes y, calculate $\prod_i P(X_i \mid Y)\, P(Y = y)$
- Choose a class y s.t. $P(Y \mid X) \propto \prod_i P(X_i \mid Y)\, P(Y = y)$ is maximal

n-gram	Spam	Ham
hello	.30	.33
friend	.08	.23
password	.28	.03
money	.40	.12

Spam	.1
Ham	.9

Spam Detection (cont'd)

An email that contains the words hello and friend, but not money and password:

- Spam: P(hello | spam) P(friend | spam) P(spam) = (.30)(.05)(.1) = 0.0015
- Ham: P(hello | ham) P(friend | ham) P(ham) = (.33)(.25)(.9) = 0.07425

n-gram	Spam	Ham
hello	.30	.33
friend	.08	.23
password	.28	.03
money	.40	.12

Spam	.1
Ham	.9

Spam Detection (cont'd)

An email that contains the words hello, money, and password:

- Spam: P(hello | spam) P(money | spam) P(password | spam) P(spam) = (.30)(.20)(.40)(.1) = 0.0024
- Ham: P(hello | ham) P(money | ham) P(password | ham) P(ham) = (.33)(0.02)(0.10)(.9) = 0.000594

n-gram	Spam	Ham
hello	.30	.33
friend	.08	.23
password	.28	.03
money	.40	.12

Spam	.1
Ham	.9

Document Clustering

- Topic modeling: assign topics (politics, sports, fashion, etc.) to documents (e.g., articles or web pages)

Document Clustering

Biomedical Articles

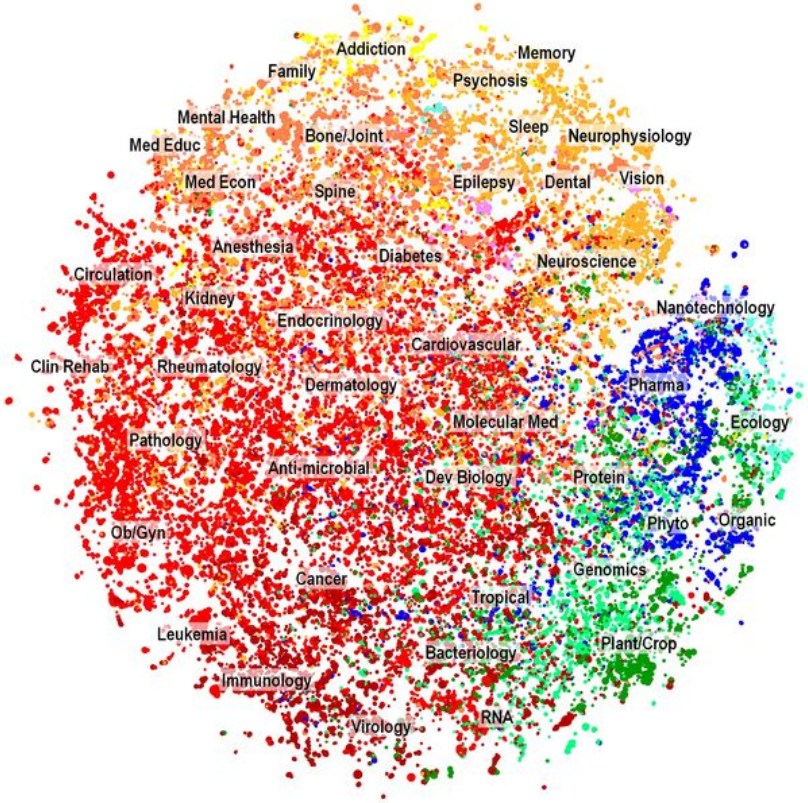


Image Source

Natural Language Generation

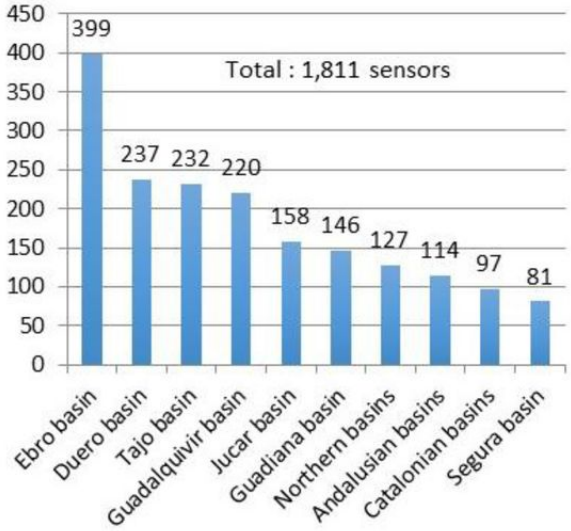
Natural Language Generation: Image Captions

Figure 5. A selection of evaluation results, grouped by human rating.

Image Source

Natural Language Generation: Descriptions

E.g., Textual descriptions of quantitative geographical and hydrological sensor data.



Flow above normal in the Jerte River at El Torno

The Jerte River at El Torno has a recorded flow of 38 m3/s, which represents an increase of 8 m3/s compared to the previous hour. The normal flow at this point of the river is 5 m3/s ...

Image Source

Natural Language Generation: Humor

Researchers developed a language model to generate jokes of the form "I like my X like I like my Y, Z"

- E.g., "I like my coffee like I like my war, cold."
- After testing, they claimed: "Our model generates funny jokes 16% of the time, compared to 33%, for human-generated jokes."

There are also language models that generate puns:

- E.g., "What do you call a spicy missile? A hot shot!"

Natural Language Understanding: Humor

Explaining a joke

Prompt

```
Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for
their TPU team? It showed them how to communicate between two
different pods!
```

Model Response

```
Prediction: TPUs are a type of computer chip that Google uses
for deep learning. A "pod" is a group of TPUs. A "pod" is also a
group of whales. The joke is that the whale is able to
communicate between two groups of whales, but the speaker is
pretending that the whale is able to communicate between two
groups of TPUs.
```

Image Source

Automatic Document Summarization

Automatically summarize documents (e.g., news articles or research papers).

Ben and Ally drove their car to go grocery shopping. They bought bananas, watermelon, and a bag of lemons and limes.

1. Extraction: copying words or phrases that are deemed interesting by some metric; often results in clumsy or grammatically-incorrect summaries:
   Ben and Ally go grocery shopping. Bought bananas, watermelon, and bag.
2. Abstraction: paraphrasing; results similar to human speech, but requires complex language modeling; active area of research at places like Google
   Ben and Ally bought fruit at the grocery store.

Sentiment Analysis

Sentiment Analysis

- Classifies a document as expressing a positive, negative, or neutral opinion.
- Especially useful for analyzing reviews (for products, restaurants, etc.) and social media posts (tweets, Facebook posts, blogs, etc.).

Twitter Data

Positive vs. Negative Words

Researchers have built lists of words with “positive” and “negative” connotations

A+
Acclaim
Accomplish
Accurate
Achievement
Admire
…

Abnormal
Abolish
Abominable
Abominate
Abort
Abrasive
…

For each chunk of our own text, we can calculate how many words lie in these “positive” or “negative” groupings

```
I love all the delicious free food in the CIT, but working in the Sunlab makes me sad.
```

We can also add common Internet slang to lists of “positive” and “negative” words: e.g, “luv”, “yay”, “ew”, “wtf”

Sentiment Analysis of Tweets

1. Download tweets from `twitter.com`
2. Preprocess tweets
   a. Remove emojis* and URLs
   b. Remove punctuation (e.g., hashtags)
   c. Split sentences into words; convert to lowercase; etc.
3. Build a language model: e.g., represent documents (i.e., tweets) by their positive, neutral, and negative words
4. Classify each tweet by scoring it based on its aggregate count: E.g., positive = +1 * count; neutral = 0; negative = -1 * count

*Comment from a student: Emojis are informative. Might do better if they are used.

Starbucks' Tweets

Using the R package `twitteR`, we can directly access Twitter data.
Here's how to access the 5000 most recent tweets about Starbucks in R:

```
library(twitteR)

starbucks_tweets <- searchTwitter('@Starbucks', n = 5000)
```

Starbucks' Tweets (cont'd)

Here's an example of 3 tweets that were returned:

```
"Wish @Starbucks would go back to fresh baked goods instead of the
pre-packaged. #sad #pastries"

“Huge shout out: exemplary service from Emile @starbucks I left
with a smile. @ Starbucks Canada https://t.co/WtXjeekCT1"

"Currently very angry at @Starbucks, for being out of their
S'mores frap at seemingly every location \xed\xa0\xbd\xed\xb8\xa1"
```

Represents

Starbucks' Tweets (cont'd)

Remove emojis: `starbucks_tweets <- iconv(starbucks_tweets, 'UTF-8', 'ASCII', sub = " ")`

Remove punctuation: `starbucks_tweets <- gsub('[[:punct:]]', ' ', starbucks_tweets)`

Remove URLs: `starbucks_tweets <- gsub('http.* *', ' ', starbucks_tweets)`

Convert to lowercase: `starbucks_tweets <- tolower(starbucks_tweets)`

"Wish @Starbucks would go back to fresh baked goods instead if the pre-packaged. #sad #pastries"

"Huge shout out: exemplary service from Emile @starbucks I left with a smile. @ Starbucks Canada https://t.co/WtXjeekCT1"

"Currently very angry at @Starbucks, for being out of their S'mores frap at seemingly every location \xed\xa0\xbd\xed\xb8\xa1"

"wish starbucks would go back to fresh baked goods instead if the prepackaged sad pastries"

"huge shout out exemplary service from emile starbucks i left with a smile starbucks canada "

"currently very angry at starbucks for being out of their smores frap at seemingly every location "

Starbucks' Tweets (cont'd)

Next, we load lists of pre-determined positive and negative words (downloaded from the Internet):

```
pos <- scan('/Downloads/positive-words.txt', what = 'character', comment.char = ';')
neg <- scan('/Downloads/negative-words.txt', what = 'character', comment.char = ';')
```

We add some informal terms of our own:

```
pos <- c(pos, 'perf', 'luv', 'yum', 'epic', 'yay')
neg <- c(neg, 'wtf', 'ew', 'yuck', 'icky')
```

Starbucks' Tweets (cont'd)

Next, we split our tweets into individual words.

```
starbucks_words = str_split(starbucks_tweets, ' ')
```

We then compare our words to the positive and negative terms.

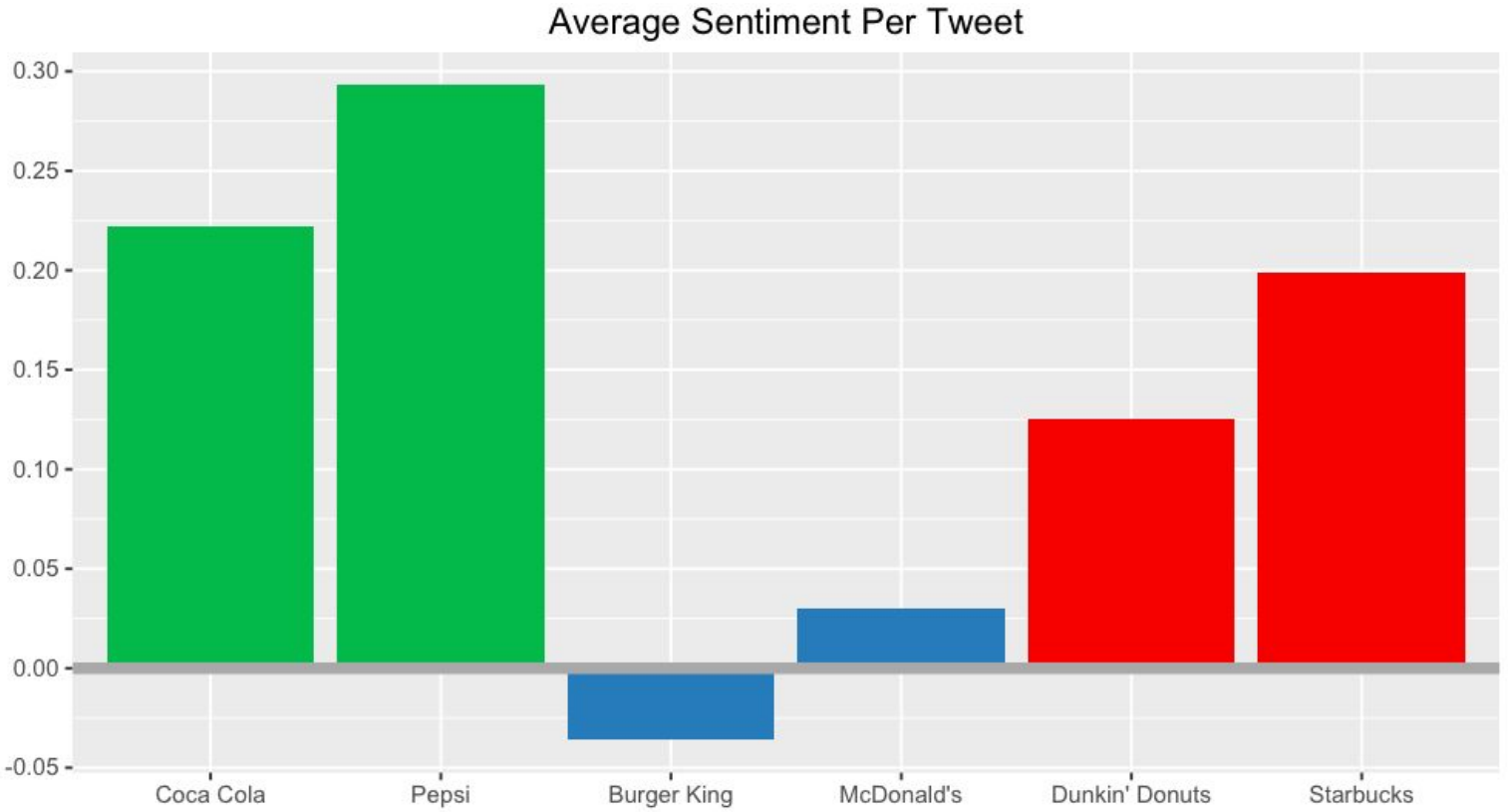
```
match(starbucks_words, pos)
match(starbucks_words, neg)
```

"wish starbucks would go back to fresh baked goods instead of the prepackaged sad pastries"
Score: 0 (*here we see limitations of this technique*)

"huge shout out exemplary service from emile starbucks i left with a smile starbucks canada"
Score: +2

"currently very angry at starbucks for being out of their smores frap at seemingly every location ”
Score: -1

Sentiment Analysis



Some Challenges

Positive words that contrast with an overall negative message (and vice versa)

"I enjoyed the old version of this app, but I hate the newer version."

Selecting the proper *N*-gram

- "This product is not reliable"
- "This product is unreliable"

If unigrams are used, "not reliable" will be split into "not" and "reliable," which could result in a neutral sentiment.

Sarcasm

"I loved having the fly in my soup! It was delicious!"

Sentiment Analysis

We can see that sentiment analysis can give a business insight into public opinion on its products and service. It can also reveal how consumers feel about a business compared to competing brands.

Businesses can also collect tweets over time, and see how sentiment changes. Can perhaps even try to build a causal model, using data about ad campaigns, new product releases, etc.

Regular Expressions

Regular Expressions (Regex)

Regular expressions are a handy tool for searching for patterns in text.

You can think of them as a fancy form of “find and replace”.

- In R, we can use `grep` to find a pattern in text:
  - `grep(regex, text)`
- And, we can use `gsub` to replace a pattern in text:
  - `gsub(regex, replacement, text)`

Regular Expressions

Consider a literature corpus, some written in American English, others in British English.

Let's find the word "color," which is equivalent to "colour."

We have a few options: e.g.,

```
grep("color|colour", text)
grep("colou?r", text)
```

`|` means "or", and `?` in this context means the preceding character is optional.

We also want to find "theater" and "theatre."

```
grep("theat(er|re)", text)
```

Regular Expressions

Next, let's find words that rhyme with "light."

```
grep("[a-zA-Z]+(ight|ite)", text)
```

`[a-zA-Z]` matches any letter

`+` matches 1 or more of the preceding characters

Tonight I might write a story about a knight with a snakebite.

```
> text <- "Tonight I might write a story about a knight with a snakebite."
> text_vec <- unlist(strsplit(text, split = "[ ]"))
> grep("[a-zA-Z]+(ight|ite)", text_vec)
[1]  1  3  4  9 12
> text_vec[grep("[a-zA-Z]+(ight|ite)", text_vec)]
[1] "Tonight"    "might"      "write"      "knight"     "snakebite."
```

Regular Expressions

Let's try numbers. For example, let's find Rhode Island zip codes. Hint: they start with 028 or 029.

```
grep("02(8|9)[0-9]{2}", text)
```

`[0-9]` matches any digit

`{2}` matches exactly 2 of the preceding characters

69 Brown Street, Providence, RI 02912

424 Bellevue Ave, Newport, RI 02840

Regular Expressions

Here's how we might we might find all instances of parents, grandparents, great grandparents, and so on.

```
grep("((great )*grand)?((fa|mo)ther)", text)
```

* captures 0 or more of the preceding characters
? in this context means the preceding expression is optional

My mother, father, grandfather, grandmother, great great grandmother, and great great great grandfather were all born in Poland.