Big Data and Economics

Tidy text toolkit

Kyle Coombs
Bates College | ECON/DCS 368

Table of contents

Prologue: Text as data

Prologue

- Today we're going be talking about text as data
    - Many resources come from Text Mining with R
- We use text all the time in our daily lives to communicate
- As a result, it is a rich source of data that can be used to answer interesting questions
- Sometimes important numerical data is embedded in text (e.g. commodity prices, wages, etc. in historical documents)
- Sometimes we need to categorize numerical data based on text (e.g. categorizing purchases based on bank memos)
- Sometimes we need to link text across datasets a "fuzzy merge" (e.g. company names, addresses, etc.)
- Sometimes the stuff we struggle to quantify is in text (e.g. sentiment, political ideology, etc.)
- Before we can get to that, we need to learn how to work with text data

Tidying text data

Tidying text data

- A library is basically a database of words
- Each word carries information
- How different words are combined together also carries information
- The problem is that text data is messy
- How could we tidy it?

Tidying text data

- There's no one structure that makes sense for all text data
- Your goal is to find a structure that makes sense for your data/research question
- Key term: **Corpus** is a collection of documents
- String variable: each row is a group of words (e.g. a sentence, title, etc.)
- Term document matrix
    - Each row is a document
    - Each column is a word
    - Each cell is the frequency of that word in that document
- Document term matrix
    - Each row is a word
    - Each column is a document
    - Each cell is the frequency of that word in that document
- You could amend the above to account for combinations of words instead of single words
- Or singleton words and groups of words (bigrams, trigrams, etc.)
- The data get big quickly!

Wider tasks with text data

- Seriously, that's a ton of words -- are they all meaningful?!

- There are lots of words in sentences and many of them are not important

- Plus words are capitalized and some are not

  - To a computer "Kyle" and "kyle" are different words
  - But to a human, they're the same word
  - But what about "Bates" and "bates"?

- Then words like "and" and "or" are called **stop words**

- Often times you'll want to remove stop words from your corpus

  - Plus, there's loads of other bits of text that you might want to remove (e.g. punctuation, numbers, etc.)
  - The package **tidytext** has a list of common stop words in `data("stop_words")`

Stop words

```r
data('stop_words')
stop_words %>% head(10)
```

```
## # A tibble: 10 × 2
##    word        lexicon
##    <chr>       <chr>
##  1 a           SMART
##  2 a's         SMART
##  3 able        SMART
##  4 about       SMART
##  5 above       SMART
##  6 according   SMART
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
```

```r
new_stop_words <- data.frame(word=c('new-stop-word','another-s
stop_words %>%
  rbind(new_stop_words) %>%
  tail(10)
```

```
## # A tibble: 10 × 2
##    word              lexicon
##    <chr>             <chr>
##  1 years             onix
##  2 yet               onix
##  3 you               onix
##  4 young             onix
##  5 younger           onix
##  6 youngest          onix
##  7 your              onix
##  8 yours             onix
##  9 new-stop-word     kyle-words
## 10 another-stop-word kyle-words
```

Wider tasks with text data

- Seriously, that's a ton of words -- are they all meaningful?!
- There are lots of words in sentences and many of them are not important
- Plus words are capitalized and some are not
  - To a computer "Kyle" and "kyle" are different words
  - But to a human, they're the same word
  - But what about "Bates" and "bates"?
- Then words like "and" and "or" are called **stop words**
- Often times you'll want to remove stop words from your corpus
  - Plus, there's loads of other bits of text that you might want to remove (e.g. punctuation, numbers, etc.)
  - The package **tidytext** has a list of common stop words in `data("stop_words")`
- But how do we remove them?! How do we identify them?

Simplest example: A string variable

- Let's say we have a database with job descriptions listed as string variables
- Look familiar?

```
## No encoding supplied: defaulting to UTF-8.

## Rows: 17070 Columns: 20
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (18): timestamp, age, industry, area, jobtitle, jobtitle2, currency, cur...
## dbl  (2): annual_salary, additional_pay
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 6 × 20
##   timestamp age   industry area  jobtitle jobtitle2 annual_salary additional_pay
##   <chr>     <chr> <chr>    <chr> <chr>    <chr>             <dbl>          <dbl>
## 1 4/11/202… 35-44 Governm… Engi… Materia… <NA>             125000            800
## 2 4/11/202… 25-34 Galleri… Gall… Assista… <NA>              71000              0
## 3 4/11/202… 35-44 Educati… Educ… Directo… <NA>              60000              0
## 4 4/11/202… 25-34 Educati… Gove… Adminis… <NA>              42000             NA
## 5 4/11/202… 18-24 Account… Admi… Executi… <NA>              65000              0
## 6 4/11/202… 25-34 Governm… Law   Counsel  <NA>              88000              0
## # ℹ 12 more variables: currency <chr>, currency_other <chr>,
## #   income_additional <chr>, country <chr>, state <chr>, city <chr>,
## #   remote <chr>, experience_overall <chr>, experience_field <chr>,
## #   education <chr>, gender <chr>, race <chr>
```

Simplest example: Matching job titles

- The job titles are free form text
- How many unique job titles are there?
- Anyone notice any issues?

```
managers2023 %>%
  group_by(jobtitle) %>%
  summarise(n=n())
```

```
## # A tibble: 9,654 × 2
##    jobtitle                                                n
##    <chr>                                               <int>
##  1 "\"Team Member, Level 1\" (retail worker)"              1
##  2 "(Junior-ish) Data Manager"                             1
##  3 "(Software) Coordinator"                                1
##  4 "(long-running community science program) Director"     1
##  5 "1st Line Support Engineer"                             1
##  6 "24/5 Live-in nanny"                                    1
##  7 "2nd Grade Teacher"                                     1
##  8 "2nd grade teacher"                                     2
##  9 "3D Artist"                                             1
## 10 "3D lab technologist"                                   1
## # ℹ 9,644 more rows
```

Case-matching the job titles

- Let's say we want to group similar job titles together
- At the very least, let's make them all lower case
- There's a lot more we could do here!

```
managers2023 %>%
  mutate(jobtitle=tolower(jobtitle)) %>%
  group_by(jobtitle) %>%
  summarise(n=n())
```

```
## # A tibble: 8,877 × 2
##    jobtitle                                              n
##    <chr>                                             <int>
##  1 "\"team member, level 1\" (retail worker)"            1
##  2 "(junior-ish) data manager"                           1
##  3 "(long-running community science program) director"   1
##  4 "(software) coordinator"                              1
##  5 "1st line support engineer"                           1
##  6 "24/5 live-in nanny"                                  1
##  7 "2nd grade teacher"                                   3
##  8 "3d artist"                                           1
##  9 "3d lab technologist"                                 1
## 10 "3rd line data engineering specialist"                1
## # ℹ 8,867 more rows
```

Ambiguous text data

- Sometimes text data is ambiguous

- For example, someone lists that they are a 24/5 live-in nanny, another says they are a live-in nanny

  - Should we group these?
  - That's a judgement call
  - Depends on the research question

- What about "Assistant Regional Manager" and "Assistant to the Regional Manager"?

- Today I'll give you the tools to implement whatever cleaning you decide

- We'll also preview ML tools to inform your decision

  - Spoiler: the more the text analysis maps to pattern recognition, the better ML will be

Dwight disagrees

Dwight Schrute would rather group them, Michael Scott would not.

Regular expressions: Swiss Army knife of

- Look at these cases where "Income - additional context" is not missing

```
## # A tibble: 5 × 1
##   income_additional
##   <chr>
## 1 Income is 70% salary, 30% commission
## 2 4% an hour retention bonus from January till September
## 3 extra money goes toward insurance
## 4 This is considered a training position. The salary is not commensurate with t…
## 5 Bonus based on work performed - usually 5-9% raise yearly as well. Hired on a…
```

- If you look at each line, you can immediately tell me what the additional pay Is
- How could we grab those paid a percentage?

Regular expressions: Swiss Army knife of

- Look at these cases where "Income - additional context" is not missing

```
## # A tibble: 5 × 1
##   income_additional
##   <chr>
## 1 Income is 70% salary, 30% commission
## 2 4% an hour retention bonus from January till September
## 3 extra money goes toward insurance
## 4 This is considered a training position. The salary is not commensurate with t…
## 5 Bonus based on work performed - usually 5-9% raise yearly as well. Hired on a…
```

- If you look at each line, you can immediately tell me what the additional pay Is
- How could we grab those paid a percentage?
- Well technically, we can go percent-by-percent!

```
managers2023 %>% select(income_additional) %>%
  mutate(ifelse('1%' %in% income_additional,1,
    ifelse('2%' %in% income_additional, 2,  ... )))
```

- This would be absurd. Do not do this unless you are participating in an International Obfuscated Code Contest

Regular expression for numbers

- Instead, we can use a regular expression to grab percentages
  - The tidyverse's own **stringr** package has a great suite of regex functions
  - There's also `grep` and `grepl` in base R, which are based on Linux's `grep` command

```
managers2023 %>% select(income_additional) %>%
  filter(!is.na(income_additional)) %>%
  mutate(add_percentage=str_extract(income_additional, '\\d+\\s*(%|percent)')) %>%
  head(5)
```

```
## # A tibble: 5 × 2
##   income_additional                                                  add_percentage
##   <chr>                                                              <chr>
## 1 Income is 70% salary, 30% commission                               70%
## 2 4% an hour retention bonus from January till September             4%
## 3 extra money goes toward insurance                                  <NA>
## 4 This is considered a training position. The salary is not comm… <NA>
## 5 Bonus based on work performed - usually 5-9% raise yearly as w… 9%
```

What is `stringr::str_extract()` doing?

- `stringr::str_extract()` is extracting the first match of a regular expression with
  - A number '\d' with at least one digit '+'
  - Followed by 0 or more spaces '\s*'
  - Followed by a percent sign '%' or the word percent
- How can we search for the '%', but not extract it and make the string numeric? Use `group`!

```
managers2023 %>% select(income_additional) %>%
  filter(!is.na(income_additional)) %>%
  mutate(add_percentage=as.numeric(str_extract(income_additional, '(\\d+)(\\s*)(%|percent)',group=1))) %>%
  head(5)
```

```
## # A tibble: 5 × 2
##   income_additional                                              add_percentage
##   <chr>                                                                   <dbl>
## 1 Income is 70% salary, 30% commission                                       70
## 2 4% an hour retention bonus from January till September                      4
## 3 extra money goes toward insurance                                          NA
## 4 This is considered a training position. The salary is not comm…            NA
## 5 Bonus based on work performed - usually 5-9% raise yearly as w…             9
```

- There's a little more clean-up needed, but that's the gist

Regular expression codes

- There are a lot of codes that you can use in regular expressions
- Here are some of the most common ones:
    - '\d' or '[0-9]' match any digit as does '[[:digit:]]' in **stringr**
    - '\D' or '[^0-9]' match any non-digit as does '[[^:digit:]]' in **stringr**
    - '\s' or '[[:space:]]'' match any whitespace character
    - '\S' or '[^[:space:]]' match any non-whitespace character
    - '\w' or '[[:word:]]' match any word character (letter, number, underscore)
    - '\W' or '[^[:word:]]' match any non-word character
    - '\b' or '\B' match word boundaries or non-word boundaries
    - '.' match any character except a newline
    - '^', '$' match the start and end of a string
    - '|' match either the expression before or after the pipe
    - '\' precedes any special character to match it literally

And many, many, many, many more

stringr functions

- There are a lot of functions in **stringr** that are useful for regular expressions
    - `str_extract()` extracts the first match
    - `str_extract_all()` extracts all matches
    - `str_detect()` detects if a string matches a pattern
    - `str_count()` counts the number of matches
    - `str_locate()` locates the position of the first match
    - `str_locate_all()` locates the position of all matches
    - `str_replace()` replaces the first match
    - `str_replace_all()` replaces all matches
    - `str_split()` splits a string into a vector of strings
    - `str_subset()` returns a subset of strings that match a pattern

And so on...

Regular expressions

- Practice makes perfect
- It takes a lot of time to get good at regular expressions
- There are fantastic tools out there, like regex101, RegExplain, **stringr** Cheatsheet
- StackOverflow is a great tool as well to see how others have solved similar problems
- Generative AI is getting better at writing regular expressions every day
- Your brain is also a critical tool for regular expressions -- and any coding task for that matter
- **Practice**: Create a regular expression that matches phone numbers in the following format: (xxx) xxx-xxxx or xxx-xxx-xxxx

    1. Create a string like
    2. Use `str_extract()` to extract the phone number

Back to the job titles

- We can create dummy variables for the job titles that mention certain words
- We can create dummy variables for job titles containing "manager" and "assistant"
- Then we can regress the salary on these dummy variables
    - I also split by remote work and cluster by industry just cause `feols()` is so neat

```
managers2023 %>%
  mutate(jobtitle=tolower(jobtitle),
      manager=str_detect(jobtitle,'manager'),
      assistant=str_detect(jobtitle,'assistant')) %>%
  feols(annual_salary ~ manager + assistant, data=.,
  fsplit=~remote, cluster=~industry) %>%
  etable()
```

```
## NOTE: 110 observations removed because of NA values (split: 64, vcov: 46).

##                                        ..1                        ..2
## Sample (remote)                Full sample               Fully remote
## Dependent Var.:              annual_salary              annual_salary
##
## Constant        119,937.2*** (13,238.5) 120,588.0*** (10,295.8)
## managerTRUE        -6,449.1 (12,907.1)      16,593.6 (14,498.2)
## assistantTRUE   -44,402.7*** (12,307.0) -42,303.0*** (10,186.2)
## _______________ _______________________ _______________________
## S.E.: Clustered            by: industry            by: industry
## Observations                     16,960                   4,344
## R2                              9.82e-5                 0.00082
## Adj. R2                        -1.97e-5                 0.00036
##
##                                        ..3                       ..4
## Sample (remote)                     Hybrid                   On-site
## Dependent Var.:              annual_salary             annual_salary
##
## Constant        138,913.9*** (26,774.6)  94,767.3*** (8,089.2)
```

Fuzzy Merge: I see a match, but the

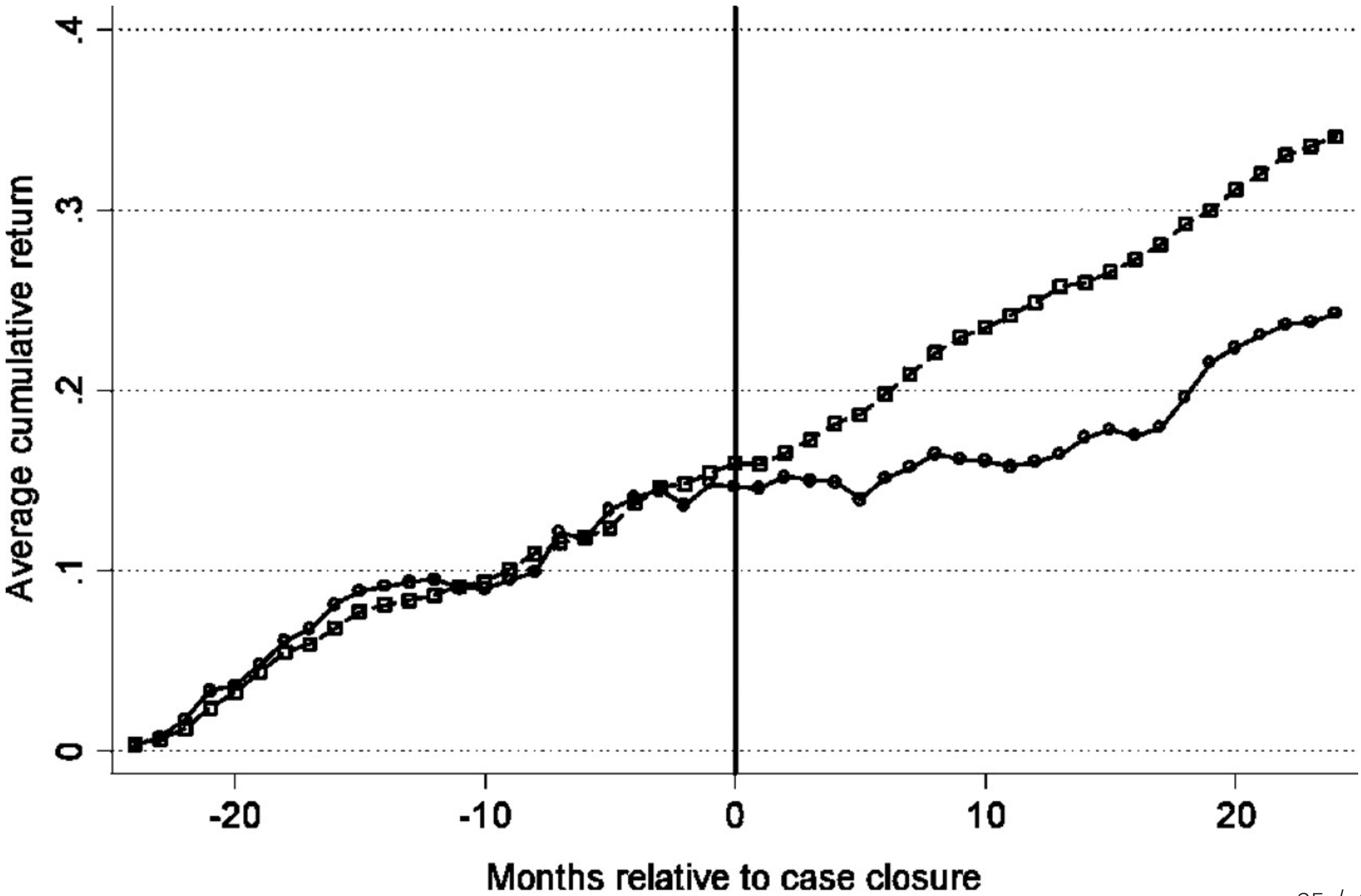
- Sometimes you have two strings that you know match, but the computer doesn't
- Before, we wanted to match job titles and we could do that by case-matching (and probably some other tricks)
- But what if there are a ton of typos? Well then we could use fuzzy matching
- Fuzzy matching is a way to match strings that are similar, but not identical
- There are a lot of ways to do this including the **stringdist**, **agrep**, and **fuzzyjoin** packages
  - True to its name **stringdist** has a suite of functions that measure the "distance" between strings
  - **fuzzyjoin** has a suite of functions that merge dataframes based on fuzzy matching
  - **agrep** is a base R function that does fuzzy matching (based on Linux) that only uses Levenshtein distance

Fuzzy match application: Union votes

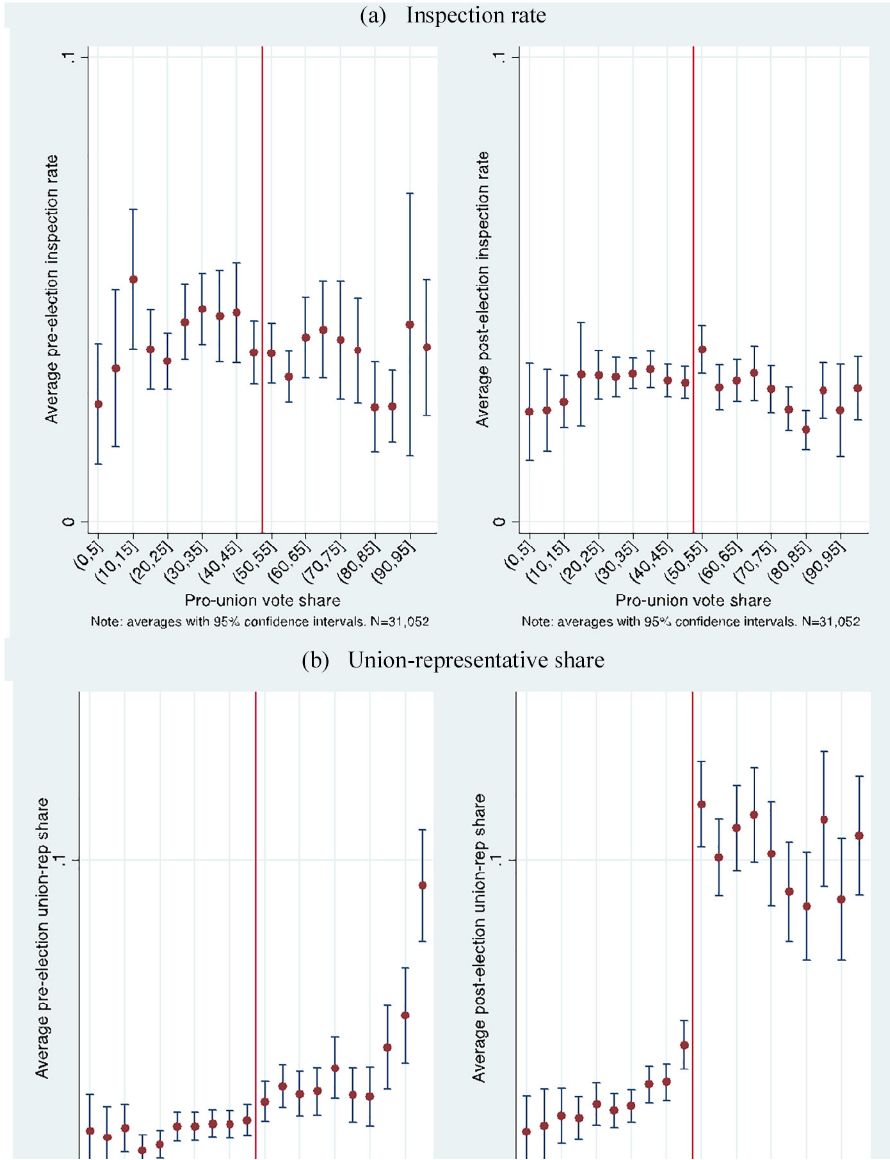
- The effect of unionization on several economic outcomes is ambiguous
  - Wages up for sure?
  - Productivity up or down?
  - Worker safety?
- The National Labor Relations Board maintains records of all labor union votes
- These records include firm name, location, vote counts, number of employees, etc.
  - No information on firm or worker outcomes
- Lee & Mas (2012) Link administrative records maintained by two separated offices:
  - NLRB union vote data + S&P Compustat firm data
  - Fuzzy match on firm name, address, etc.
  - Long-run event studies show a 10% decline in equity value of firm after union vote
  - Cannot decompose into wage premia and productivity change
- Sojourner & Yang (2022) link to Occupational Safety and Health Administration data
  - OSHA inspection increases after union vote, more violations cited and penalties assessed

(Recent work shows bias against unions when Republicans control NLRB compromising the validity of all union RDD results)

Firm Cumulative Absolute Return



OSHA Inspections



Distance between strings?

- What does it mean to measure the distance between strings?
- Well, we can think of strings as vectors or groups of characters
- Think of the distance between strings as the changes between these characters
- **Levenshtein**: Measure number of characters missing, added, or substituted
  - "Kyle" and "Kile" have a Levenshtein distance of 1
  - "Kyle" and "Klye" have a Levenshtein distance of 2
- We can account for transpositions as well (Damerau-Levenshtein distance)
  - "Kyle" and "Klye" have a Damerau-Levenshtein distance of 1
- There are many other distance measures (Jaro-Winkler, Hamming, Phonetic, etc.)
- Normalize the distance by the length of the string to get a measure of similarity
- If the similarity exceeds a threshold you choose, we can say that the strings match

String distance

Mock Harry Potter dataset examples from R-Vogg-Blog

```
stringdistmatrix(input,compare,
  method = "lv",
  useNames = "strings")
##                Harry Potter Voldemort
## harry j potter            4        12
## harrypotter               3         9
## Voldemort                10         0
## Harry POTTER              5        12
## Harrry Potter             1        11
## Ron Weasley              11         9
```

```
tidy_comb(input,compare[1]) %>%
  tidy_stringdist(method=c('lv','dl','jw','cosine')) %>%
  rename(Levenshtein=lv,`Damerau-Levenshtein`=dl,`Jaro-Winkler`=j
## # A tibble: 6 × 6
##   V1           V2         Levenshtein `Damerau-Levenshtein` `J
## * <chr>        <chr>            <dbl>                 <dbl>
## 1 Harry Potter harry j …            4                     4
## 2 Harry Potter harrypot…            3                     3
## 3 Harry Potter Voldemort           10                    10
## 4 Harry Potter Harry PO…            5                     5
## 5 Harry Potter Harrry P…            1                     1
## 6 Harry Potter Ron Weas…           11                    11
```

Fuzzy matching to merge

```r
fuzzyjoin::stringdist_join(df1, df2,
                mode = "inner",
                by = "name",
                max_dist = 6,
                method='lv')
```

```
##            name.x       name.y bad_spells_index
## 1 harry j potter Harry Potter             0.02
## 2     harrypotter Harry Potter             0.02
## 3       Voldemort    Voldemort             0.87
## 4    Harry POTTER Harry Potter             0.02
## 5   Harrry Potter Harry Potter             0.02
```

```r
fuzzyjoin::stringdist_join(df1, df2,
                mode = "inner",
                by = "name",
                max_dist = 10,
                method='lv')
```

```
##            name.x       name.y bad_spells_index
## 1 harry j potter Harry Potter             0.02
## 2     harrypotter Harry Potter             0.02
## 3     harrypotter    Voldemort             0.87
## 4       Voldemort Harry Potter             0.02
## 5       Voldemort    Voldemort             0.87
## 6    Harry POTTER Harry Potter             0.02
## 7   Harrry Potter Harry Potter             0.02
## 8     Ron Weasley    Voldemort             0.87
```

Fuzzy matching to group rows

- Can be done with `tidystringdist`, but it gets slow fast (lots of comparisons!)
- Could parallelize comparisons to speed it up, but you need to write the code yourself

```
managers2023 %>%
  head(1000) %>%
  distinct(jobtitle) %>%
  tidy_comb_all(jobtitle) %>%
  tidy_stringdist() %>%
  filter(lv <= 1) # at most 1 character difference
```

```
## # A tibble: 6 × 12
##   V1           V2      osa    lv    dl hamming   lcs qgram cosine jaccard     jw
##   <chr>        <chr> <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>  <dbl>   <dbl>  <dbl>
## 1 grant admin… gran…     1     1     1     Inf     1     1 0.0111  0      0.0868
## 2 vice presid… vice…     1     1     1       1     2     2 0.0455  0.167  0.0476
## 3 coo          cto       1     1     1       1     2     2 0.225   0.333  0.222
## 4 pr manager   hr m…     1     1     1       1     2     2 0.0714  0.222  0.0667
## 5 operations … oper…     1     1     1     Inf     1     1 0.0157  0.0833 0.0185
## 6 solution ar… solu…     1     1     1     Inf     1     1 0.0148  0      0.0175
## # ℹ 1 more variable: soundex <dbl>
```

Practical advice on fuzzy matching

- Fuzzy matching is a great tool, but it's not magic
  - It can also lie to you
- Don't use it when you:
  - Have a reliable key/id between two dataframes
  - Can easily clean the data to make a reliable key/id
- Match on any many:many keys, then fuzzy match within the group to get the best unique link
- Use it to create a reliable key once that you can then reuse rather than re-running
  - This helps both stability, reproducibility, and speed
- It is as much an art as it is a science
  - You'll need to make judgement calls about what is a match and what is not
  - You'll need to make judgement calls about what distance threshold to use
  - You'll need to make judgement calls about what distance measure to use
- More than likely you'll get false positives and negatives in any given fuzzy merge
  - LLMs have made strides in putting some structure on this, but it's still an art
  - (One day this skill might be obsolete though)

Fuzzy match guidelines

When to fuzzy match

- Too much data to hand match (large N)
- No reliable key/id
- You can't clean the data to make a reliable key/id

When not to fuzzy match

- You can match manually (small N)
- You have a reliable key/id
- You can clean the data to make a reliable key/id

No unique match even after fuzzy match?

- Perform analysis on the group of matches, on each individual match, etc. to see how sensitive your results are to each match

- Hopefully, a mismatch is "classical measurement error," which is an endogeneity problem that puts a downward bias on results

Summarizing text

Summarizing text

- There are a lot of ways to summarize text
- We'll focus on three today:
    - Word counts: How many words are there?
    - Word clouds: Let's see them all together
    - Sentiment analysis: How positive or negative is the text?
- None of these are machine learning tools, but they can be used to inform machine learning tools
    - For example, word counts can be used to create a term document matrix for topic modeling
- They're also useful for exploratory data analysis
- But don't mistake them for the cutting edge analysis
    - Especially sentiment analysis, which is a very blunt tool
    - But it is a bridge to topic modeling and other NLP tools

Word counts: Term frequency

- Word counts are the simplest way to summarize text
- Literally just count up the number of words
- We can do this manually, or we can use the **tidytext** package function `unnest_tokens()`
    - `unnest_tokens()` splits a string variable into a new row for each "token"
    - Then you can count

```
tokens <- managers2023 %>%
  select(jobtitle) %>%
  filter(!is.na(jobtitle)) %>%
  mutate(jobtitle=tolower(jobtitle)) %>%
  unnest_tokens(word,jobtitle) %>%
  count(word,sort=T)
tokens
```

```
## # A tibble: 2,384 × 2
##    word            n
##    <chr>       <int>
##  1 manager      3483
##  2 senior       1924
##  3 director     1856
##  4 engineer     1088
##  5 of            976
##  6 assistant     945
##  7 analyst       916
##  8 specialist    852
##  9 associate     800
## 10 coordinator   662
## # ℹ 2,374 more rows
```

Stop words

- Did you notice that "of" was one of the most common words?
- It is in a lot of job titles, but it's not very informative
- Imagine if this weren't job titles, but a corpus of text from a novel
  - You'd be constantly panning for "gold" words amidst a see of "of"s and "the"s
- Let's get rid of it using the **tidytext** package's `stop_words` dataset and `anti_join()`

```
data('stop_words')
tokens_no_stops <- tokens %>%
  anti_join(stop_words)
```

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

```
tokens_no_stops
```

```
## # A tibble: 2,290 × 2
##    word            n
##    <chr>       <int>
##  1 manager      3483
##  2 senior       1924
##  3 director     1856
##  4 engineer     1088
##  5 assistant     945
##  6 analyst       916
##  7 specialist    852
##  8 associate     800
##  9 coordinator   662
## 10 software      602
```

Word Cloud

- Word clouds are a great way to visualize word counts
- The size of the word is proportional to the number of times it appears

```
pal <- brewer.pal(8,"Dark2") # define a nice color palette with function from RColorBrewer

tokens_no_stops %>%
  with(wordcloud(word, n, random.order = FALSE, max.words = 50, colors=pal))
```

n-grams: phrases

- Sometimes words often go together
- For example, "machine learning" is a phrase
- If we just count the mentions of "machine" and "learning" separately, we lose the context
- We can use the **tidytext** package's `unnest_tokens()` function to create n-grams
- **ngram** literally means give me all groups of "n words"

In practice

Bigrams will count a single word in multiple bigrams:

- "a machine learning algorithm" will count "a machine," "machine learning," and "learning algorithm"

```
bigrams <- managers2023 %>%
  select(jobtitle) %>%
  filter(!is.na(jobtitle)) %>%
  mutate(jobtitle=tolower(jobtitle)) %>%
  unnest_tokens(word,jobtitle,token='ngrams',n=2) %>%
  count(word,sort=T)
bigrams
```

```
## # A tibble: 9,769 × 2
##    word                  n
##    <chr>             <int>
##  1 <NA>               1691
##  2 director of         651
##  3 software engineer   410
##  4 project manager     341
##  5 program manager     243
##  6 senior software     187
##  7 associate director  173
##  8 human resources     152
##  9 senior manager      150
## 10 operations manager  147
## # ℹ 9,759 more rows
```

Separate out n-grams, remove stop

```r
 bigrams_separated <- bigrams %>%
  separate(word,c('word1','word2'),sep=" ")
bigrams_separated
```

```
## # A tibble: 9,769 × 3
##    word1      word2          n
##    <chr>      <chr>      <int>
##  1 <NA>       <NA>        1691
##  2 director   of           651
##  3 software   engineer     410
##  4 project    manager      341
##  5 program    manager      243
##  6 senior     software     187
##  7 associate  director     173
##  8 human      resources    152
##  9 senior     manager      150
## 10 operations manager      147
## # ℹ 9,759 more rows
```

```r
bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
```

```
## # A tibble: 8,292 × 3
##    word1      word2          n
##    <chr>      <chr>      <int>
##  1 <NA>       <NA>        1691
##  2 software   engineer     410
##  3 project    manager      341
##  4 program    manager      243
##  5 senior     software     187
##  6 associate  director     173
##  7 human      resources    152
##  8 senior     manager      150
##  9 operations manager      147
## 10 vice       president    132
## # ℹ 8,282 more rows
```

Term frequency-inverse doc frequency

- Frequencies are useful, but they don't tell us much about the context of the words
- We need to know how unique a word is to a document
- Effectively, a document is a group of words (e.g. a sentence, a job title, an essay, etc.)
- A term is a word/phrase
- Some words are uniquely common to a "document" (e.g. "manager" in a job title)
- So they may be valuable to predicting/classifying something about that "document" (e.g. salary, industry)

Term frequency-inverse doc frequency

- Term frequency is the number of times a term appears in a document divided by the total number of terms in the document
- Inverse document frequency of a term is the log of the number of documents divided by the number of documents containing ta term

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

- **Note**: This is a heuristic with many variations and shaky theoretical foundations
- Roughly, the more documents a term appears in, the less valuable it is to predicting/classifying something about that document
- As such, the idf falls as the number of documents containing a term increases
- The $tf - idf$ is the product of the term frequency and the inverse document frequency

Where is this all headed?

- We can use the $tf - idf$ to predict the industry of a job title, the topics of a book, content of a tweet, etc.
- We have a bunch of job titles categorized by industry, salary, etc.
- We could use the $tf - idf$ to predict the industry or salary of a new job title
- Alternatively, say we have a bunch of tweets and we want to know if they are positive or negative
  - We could search for a bunch of terms OR we could flag several thousand tweets as positive or negative
  - Then we could feed the text to a machine learning algorithm that uses the $tf - idf$ to infer whether a word, its common ngrams, etc. are positive or negative
  - Then it could predict the sentiment of new tweets
- This is the basic idea behind topic modeling and sentiment analysis and how we get to GPT-4

Next lecture: Sentiment analysis, basics of topic modeling