Topic 11: Text Analytics

ISOM3390: Business Programming in R

Basic Workflow for Text Analytics

- Obtain the text sources
- Extract documents and move into a corpus
- Transformation. This typically involves:
    - Case folding - usually convert to lower case
    - Punctuation and number removals
    - Stop word removal - common words that are not informative as "the", "of", "to", and so forth in English
    - Stemming - reduce words to their word stem, e.g., "Fishing", "fished", and "fisher" -> "fish"
- Extract features - convert the text string into some sort of quantifiable measures
- Perform analysis - e.g., text classification, topic modeling, etc.

Tidy Text Format and `tidytext`

Mirroring tidy data principles, the **tidy text format** is defined as:

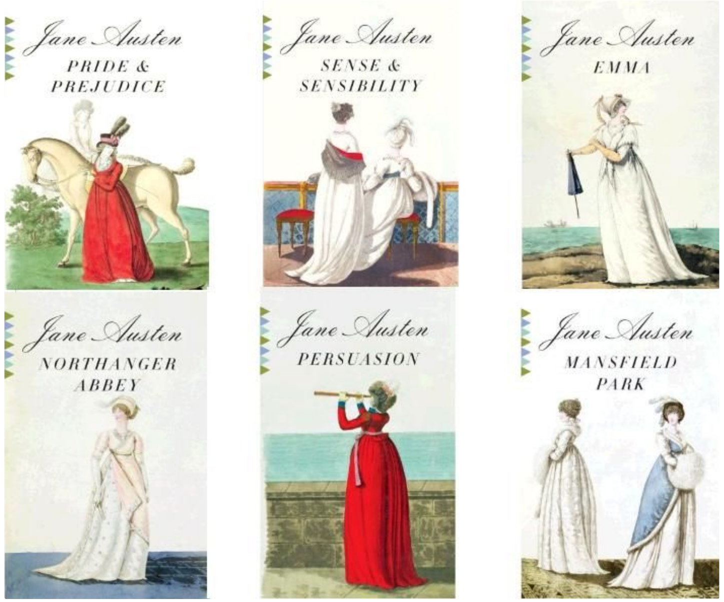
- Being a table with one term per row.
- Differs from the **document-term matrix** that is one-document-per-row and one-term-per-column.

Arranging text in the tidy text format allows us to use the `tidyverse` to explore and visualize text data coherently.

The `tidytext` package provides functions and supporting datasets to allow conversion of text to and from tidy formats, and to switch seamlessly between tidy tools and existing text mining packages.

```
library(tidytext)
```

janeausten r



The `janeaustenr` package provides the text of Jane Austen's 6 published novels in a one-row-per-line format. A function `austen_books()` returns a tidy data frame of all 6 novels.

```
library(janeaustenr)
```

Indexing and Annotating Lines

Use `mutate()` in `dplyr` to annotate each line with its line number and chapter:

```
original_books <- austen_books() %>% group_by(book) %>% mutate(linenumber = row_number(),
    chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]", ignore_case = TRUE)))) %>%
    ungroup()
```

Note: `regex()` is a `stringr`'s modifier function that controls the actual matching behavior defined by a **regexp**.

Tokenizing Text

`unnest_tokens()` in `tidytext` tokenizes the text by commonly used units of text (also removes all tokens which are strictly numbers):

```
str(unnest_tokens)
```

```
## function (tbl, output, input, token = "words", format = c("text", "man", "latex", "html", "xml"), to_lower = TRUE,
##     drop = TRUE, collapse = NULL, ...)
```

```
tidy_books <- original_books %>% unnest_tokens(word, text) %>% filter(!str_detect(word, "^[0-9]*$"))
```

`unnest_token` function also strips all punctuation, and converts each word to lowercase for easy comparability.

Other options for `token` include characters, n-grams, sentences, paragraphs, separation around a **regular expression**, ect.

Tokenize text into sentences:

```
data_frame(text = prideprejudice) %>% unnest_tokens(sentence, text, token = "sentences") %>% .$sentence %>% .[4]
```

```
## [1] "bennet,\" said his lady to him one day, \"have you heard that netherfield park is let at last?\""
```

Split text using a **regexp**:

```
austen_books() %>% group_by(book) %>% unnest_tokens(chapter, text, token = "regex",
                                                    pattern = "Chapter|CHAPTER [\\dIVXLC]") %>% ungroup()
```

```
## # A tibble: 275 x 2
##    book                chapter
##    <fct>               <chr>
##  1 Sense & Sensibil… "sense and sensibility\n\nby jane austen\n\n(1811)\n\n\n\n\n"
##  2 Sense & Sensibil… "\n\n\nthe family of dashwood had long been settled in sussex.  their estate\nwas large, and thei…
##  3 Sense & Sensibil… "\n\n\nmrs. john dashwood now installed herself mistress of norland; and her\nmother and sisters-…
##  4 Sense & Sensibil… "\n\n\nmrs. dashwood remained at norland several months; not from any\ndisinclination to move whe…
##  5 Sense & Sensibil… "\n\n\n\"what a pity it is, elinor,\" said marianne, \"that edward should have no\ntaste for draw…
##  6 Sense & Sensibil… "\n\n\nno sooner was her answer dispatched, than mrs. dashwood indulged herself\nin the pleasure …
##  7 Sense & Sensibil… "\n\n\nthe first part of their journey was performed in too melancholy a\ndisposition to be other…
##  8 Sense & Sensibil… "\n\n\nbarton park was about half a mile from the cottage.  the ladies had\npassed near it in the…
##  9 Sense & Sensibil… "\n\n\nmrs. jennings was a widow with an ample jointure.  she had only two\ndaughters, both of wh…
## 10 Sense & Sensibil… "\n\n\nthe dashwoods were now settled at barton with tolerable comfort to\nthemselves.  the house…
## # ... with 265 more rows
```

Removing Stop Words

`tidytext` has a dataset named `stop_words` for English stop words.

Use `anti_join()` or `filter()` to remove them from further analysis:

```
tidy_books <- tidy_books %>% anti_join(stop_words)
```

Now the data is ready for some basic analyses (e.g., use `dplyr`'s `count()` to find the most common words) with the data.

Sentiment Analysis

Human readers use their understandings of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterize it by some more nuanced emotion.

The Favourite (2018)

★ 10/10

I was left shaking after watching this film

5 November 2018

Honestly all I can say is that this film was not what I was expecting and far exceeded my expectations. The chemistry between the actors and also the visual story is absolutely stunning and I'm just wowed by how well done everything is done in this film. I can't say I have anything bad to say about this film. And please go into this movie without spoilers, I find that it is way more enjoyable to be surprised by the actual story and leaves more excitement for the viewer.

La La Land (2016)

★ 4/10

average, but keep trying Hollywood

16 January 2017

The dancing was average at best. The opening scene was the best routine. These actors are not Ginger and Fred. I am not sure if the talent was missing, or something else. The singing was either weak, or mixed poorly. The song writing was unique, but not strong enough to remember. The story line was predictable but sweet. I was so looking forward to seeing a musical movie. It has received great reviews because I believe those reviewers were all so desperate to see an upbeat, musical love story, as was I. So, in conclusion, it is worth seeing, if only to just to send Hollywood a message, with your dollars. More of this but better quality please.

Sentiment Lexicons

List-based sentiment analysis draws upon positive and negative word sets (called **sentiment lexicons or dictionaries**) that convey human emotion or feeling.

The `tidytext` package tabulates several sentiment lexicons in the `sentiments` dataset.

`tidytext` provides the function `get_sentiments()` to get individual sentiment lexicons.

List-Based Sentiment Scores

Sentiment analysis is measurement-focused.

We can draw upon a lexicon to identify a list of positive and negative words, and count their numbers to derive sentiment scores for text:

```
(tidy_books_bing <- tidy_books %>% inner_join(get_sentiments("bing")))
```

```
## # A tibble: 44,171 x 5
##    book                linenumber chapter word        sentiment
##    <fct>                    <int>   <int> <chr>       <chr>
##  1 Sense & Sensibility         16       1 respectable positive
##  2 Sense & Sensibility         18       1 advanced    positive
##  3 Sense & Sensibility         20       1 death       negative
##  4 Sense & Sensibility         21       1 loss        negative
##  5 Sense & Sensibility         25       1 comfortably positive
##  6 Sense & Sensibility         28       1 goodness    positive
##  7 Sense & Sensibility         28       1 solid       positive
##  8 Sense & Sensibility         29       1 comfort     positive
##  9 Sense & Sensibility         30       1 relish      positive
## 10 Sense & Sensibility         33       1 steady      positive
## # ... with 44,161 more rows
```

Defining the Context to Measure Sentiments

The size of the chunk of text that we use to add up sentiment scores can have an effect on an analysis.

Here we use integer division (`%/%`) to define larger sections of text that span 80 lines.

```
tidy_books_bing %>% count(book, index = linenumber %/% 80, sentiment)
```

```
## # A tibble: 1,840 x 4
##    book                index sentiment     n
##    <fct>               <dbl> <chr>     <int>
##  1 Sense & Sensibility     0 negative     16
##  2 Sense & Sensibility     0 positive     26
##  3 Sense & Sensibility     1 negative     19
##  4 Sense & Sensibility     1 positive     44
##  5 Sense & Sensibility     2 negative     12
##  6 Sense & Sensibility     2 positive     23
##  7 Sense & Sensibility     3 negative     15
##  8 Sense & Sensibility     3 positive     22
##  9 Sense & Sensibility     4 negative     16
## 10 Sense & Sensibility     4 positive     29
## # ... with 1,830 more rows
```

Computing Simple-Difference Sentiment Scores

Use `spread` in `tidyr` to reshape the data and compute difference scores:

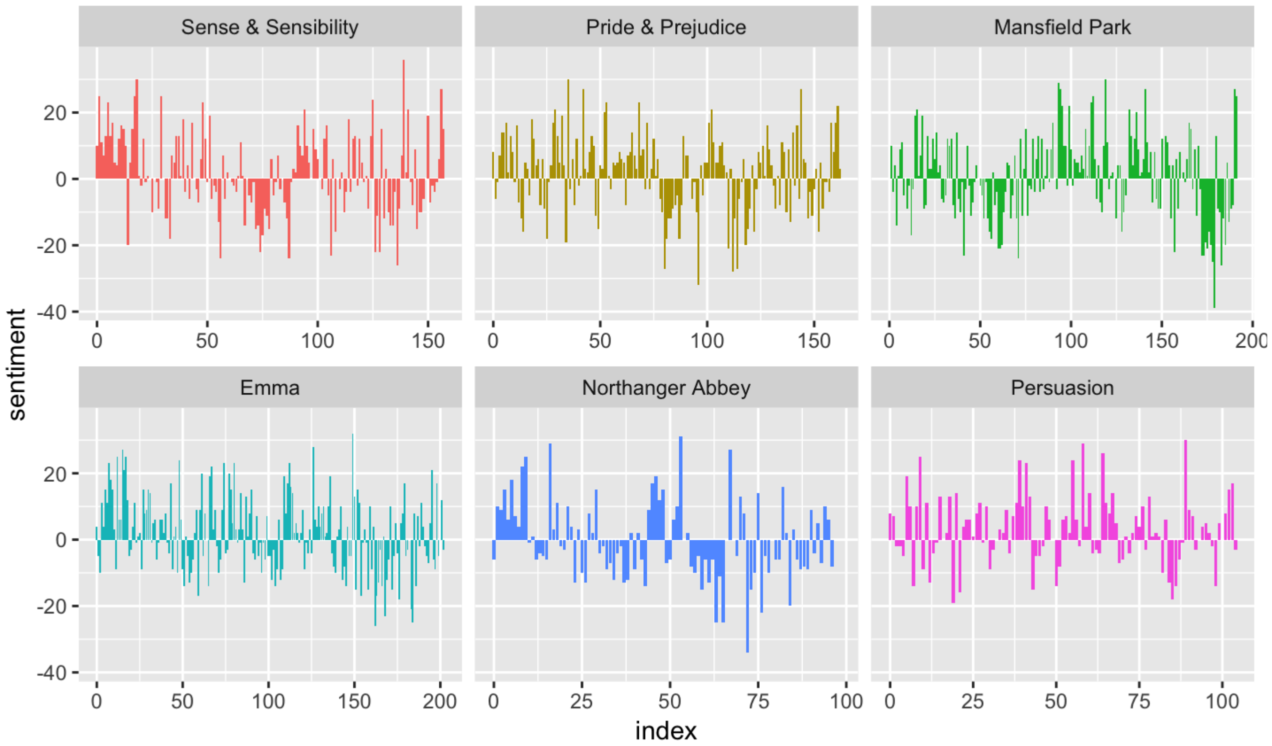
```
(sentiment_bing <- tidy_books_bing %>% count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>% mutate(sentiment = positive - negative))
```

```
## # A tibble: 920 x 5
##    book                index negative positive sentiment
##    <fct>               <dbl>    <dbl>    <dbl>     <dbl>
##  1 Sense & Sensibility     0       16       26        10
##  2 Sense & Sensibility     1       19       44        25
##  3 Sense & Sensibility     2       12       23        11
##  4 Sense & Sensibility     3       15       22         7
##  5 Sense & Sensibility     4       16       29        13
##  6 Sense & Sensibility     5       16       39        23
##  7 Sense & Sensibility     6       24       37        13
##  8 Sense & Sensibility     7       22       39        17
##  9 Sense & Sensibility     8       30       35         5
## 10 Sense & Sensibility     9       14       18         4
## # ... with 910 more rows
```

To predict whether a section posesses a positive or negative sentiment (i.e., do classification), we can use a training-set-developed cutoff when we have labeled sections.

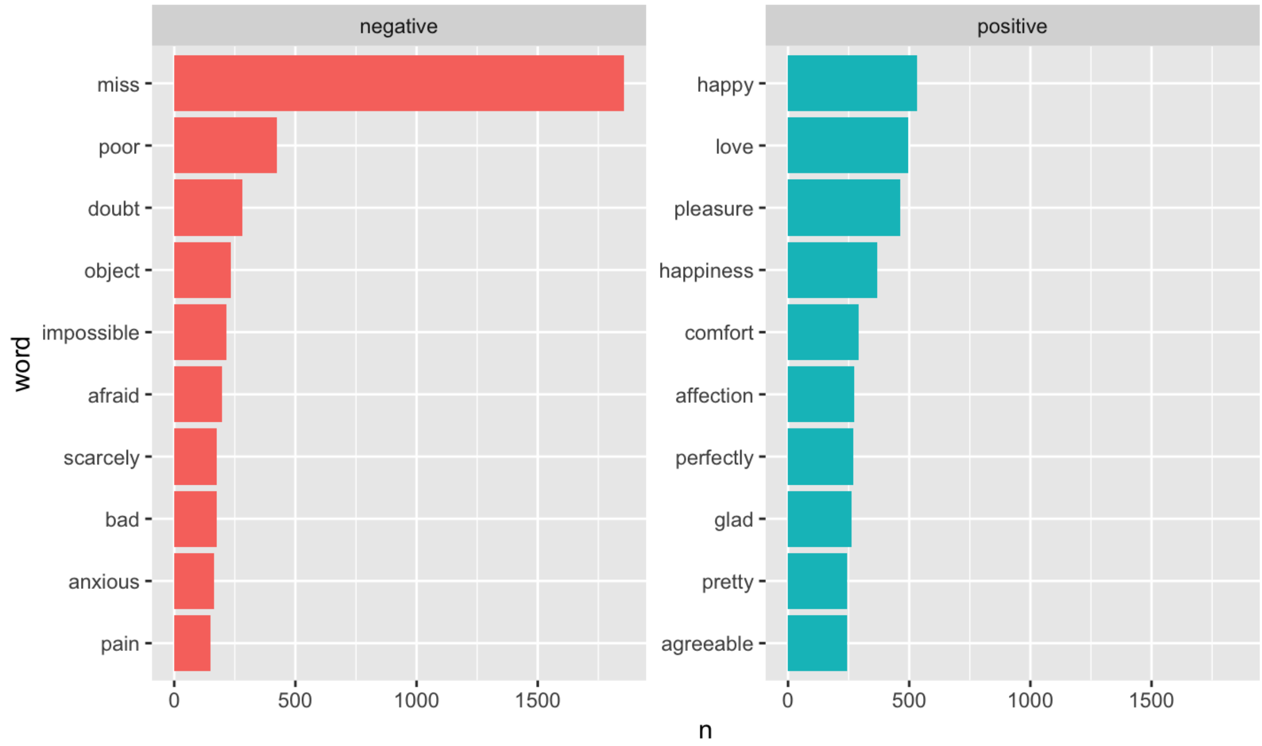
Ploting Sentiments

```
sentiment_bing %>% ggplot(aes(index, sentiment, fill = book)) + geom_col(show.legend = FALSE) +
facet_wrap(~ book, ncol = 3, scales = "free_x")
```



Finding Most Common Words

```
tidy_books_bing %>% count(word, sentiment) %>% ungroup() %>%
  group_by(sentiment) %>% top_n(10) %>% ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) + geom_col(show.legend = FALSE) + facet_wrap(~ sentiment, scales = "free_y") + coord_flip()
```



Wordclouds

Another way to visualize word frequencies:

```r
library(wordcloud)
tidy_books %>% count(word) %>%
      with(wordcloud(word, n, min.freq = 20, max.words = 100, rot.per = 0.35, colors = brewer.pal(6, "Dark2")))
```

Tag positive and negative words using `comparison.cloud`, which receives a matrix whose rows represent words and whose columns represent setiments:

```
tidy_books_bing %>% count(word, sentiment) %>% ungroup() %>% spread(sentiment, n, fill = 0) %>%
  remove_rownames() %>% column_to_rownames("word") %>% as.matrix() %>%
  comparison.cloud(colors=c("#F8766D", "#00BFC4"), max.words = 100)
```

```
##            negative positive
## abominable       17        0
## abominably        7        0
## abominate         3        0
## abound            0        1
## abrupt            5        0
## abruptly         12        0
## absence         111        0
## absurd           19        0
## absurdity        12        0
## abundance         0       14
## abundant          0        2
## abuse             8        0
## abused            6        0
## abuses            1        0
## abusive           2        0
## abyss             1        0
## accessible        0        1
## accidental       11        0
```

Critique for List-Based Text Measures

Movie	Total Words	Positive Words	Negative Words	Text Measures POSITIVE	Text Measures NEGATIVE	Rating	Thumbs Up/Down
Marigolds	26	0	1	0.00	3.85	10	UP
Blade Runner	21	2	0	9.52	0.00	9	UP
Vinny	29	1	2	3.45	6.90	4	DOWN
Mars Attacks	20	1	0	5.00	0.00	7	UP
Fight Club	18	0	2	0.00	11.11	2	DOWN
Congeniality	10	0	1	0.00	10.00	1	DOWN
Find Me Guilty	18	0	2	0.00	11.11	7	UP
Moneyball	36	2	1	5.56	2.78	4	DOWN

We could review a bad movie using words chosen from the positive list or a good movie using words chosen from the negative list.

There is nothing inherently good about the positive words or inherently bad about the negative words. It is **context** that gives them meaning.

Context Matters

Many interesting text analyses are based on the relationships between words, for example

- Which words tend to follow others immediately, e.g.:
  - The words "happy" and "like" in a sentence like "I'm not happy and I don't like it!".
  - The word "unpredictable" in a phrase "unpredictable steering" in an automotive review vs. in a phrase "unpredictable plot" in a movie review.
- Which words tend to co-occur within the same documents, e.g.:
  - The word “lead" in "solid wastes in the lead industry are potentially hazardous" vs. in "technological advancements in electrical equipment, metal and more have continued to lead industry trends in both productivity and sustainability".

Sequences of consecutive words could provide contexts, while **modeling techniques** could learn about them.

Tokenizing by N-gram

Capturing relationships between words require tokenizing by sequences of adjacent words, called **n-grams**.

`unnest_tokens` can tokenize text into **n-grams** using the `token = "ngrams"` option and setting `n` to the number of words we wish to capture in each n-gram.

```
austen_bigrams <- austen_books() %>% unnest_tokens(bigram, text, token = "ngrams", n = 2) %>% separate(bigram, c("word1",
    "word2"), sep = " ")
```

The most common bigrams are pairs of common (uninteresting) words (stop words), such as "of the" and "to be".

Filtering N-grams

Use `tidyr`'s `separate()` to separate bigrams into their constituents, remove cases where either is a stop word, and recombine them into one:

```
(bigrams_count <- austen_bigrams %>% filter(!word1 %in% stop_words$word) %>%
   filter(!word2 %in% stop_words$word) %>%
   count(book, word1, word2, sort = TRUE) %>% unite(bigram, word1, word2, sep = " "))
```

```
## # A tibble: 36,217 x 3
##    book                bigram                n
##    <fct>               <chr>             <int>
##  1 Mansfield Park      sir thomas          287
##  2 Mansfield Park      miss crawford       215
##  3 Persuasion          captain wentworth   170
##  4 Emma                miss woodhouse      162
##  5 Emma                frank churchill     132
##  6 Persuasion          lady russell        118
##  7 Mansfield Park      lady bertram        114
##  8 Persuasion          sir walter          113
##  9 Emma                miss fairfax        109
## 10 Sense & Sensibility colonel brandon     108
## # ... with 36,207 more rows
```

"separate/filter/count/unite" lets us find the most common bigrams not containing stop words.

Negating Words

Examine how often sentiment-associated words are preceded by negating words:

```
(neg_bigrams <- austen_bigrams %>%
   filter(word1 %in% c("not", "no", "never", "without")) %>%
   inner_join(get_sentiments("afinn"), by = c(word2 = "word")) %>%
   count(word1, word2, score, sort = TRUE) %>% ungroup()) %>%
  mutate(score = - score) %>% unite(bigram, word1, word2, sep = " ")
```

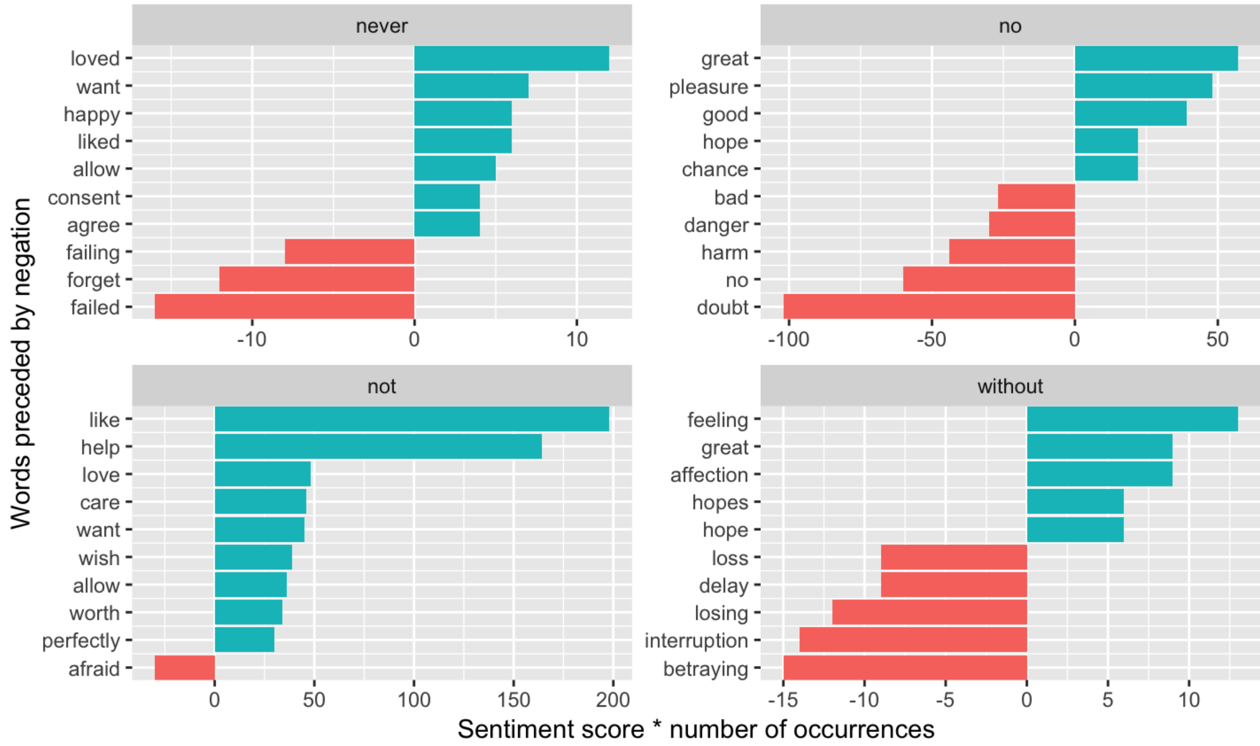
```
## # A tibble: 531 x 3
##    bigram    score     n
##    <chr>     <int> <int>
##  1 no doubt      1   102
##  2 not like     -2    99
##  3 not help     -2    82
##  4 no no         1    60
##  5 not want     -1    45
##  6 not wish     -1    39
##  7 not allow    -1    36
##  8 not care     -2    23
##  9 no harm       2    22
## 10 not sorry     1    21
## # ... with 521 more rows
```

```
(neg_bigrams <- neg_bigrams %>%
   mutate(contribution = n * score) %>%
   arrange(desc(abs(contribution))) %>%
   split(.$word1) %>% map_dfr(~ head(.x, n = 10)) %>%
   arrange(word1, contribution) %>% mutate(order = row_number()))
```

```
## # A tibble: 40 x 6
##    word1 word2    score     n contribution order
##    <chr> <chr>    <int> <int>        <int> <int>
##  1 never failed      -2     8          -16     1
##  2 never forget      -1    12          -12     2
##  3 never failing     -2     4           -8     3
##  4 never agree        1     4            4     4
##  5 never consent      2     2            4     5
##  6 never allow        1     5            5     6
##  7 never liked        2     3            6     7
##  8 never happy        3     2            6     8
##  9 never want         1     7            7     9
## 10 never loved        3     4           12    10
## # ... with 30 more rows
```

`purrr::map_dfr()` in `purrr` transform their input by applying a function to each element and returning a data frame created by row-binding.

```r
neg_bigrams %>% ggplot(aes(order, contribution, fill = n * score > 0)) + geom_bar(stat = "identity", show.legend = FALSE) +
    facet_wrap(~word1, scales = "free") + xlab("Words preceded by negation") + ylab("Sentiment score * number of occurrences") +
    scale_x_continuous(breaks = neg_bigrams$order, labels = neg_bigrams$word2, expand = c(0, 0)) + coord_flip()
```



Text Classification

A **text classification problem** may be addressed using various techniques:

- Supervised: naive Bayes, k-nearest neighbors, logistic regression, random forests, support vector machines, …
- Unsupervised: clustering, topic modeling, …

The basic process for supervised learning is:

- Train a model on the labeled data (sometimes require hand-coding), using the variable as the outcome of interest and the text features of the documents as the predictors;
- Evaluate the effectiveness of the statistical learning model via a resampling method (e.g., cross-validation);
- Apply the model to the remaining set of documents that have never been labeled.

USCongress

`USCongress` in `RTextTools` is a sample dataset containing labeled bills from the United States Congress:

```
library(RTextTools)
data(USCongress)
congress <- as_tibble(USCongress) %>% mutate(text = as.character(text))
```

Text Processing for a Tidy Text Data Frame

```
(congress_tokens <- congress %>% unnest_tokens(word, text) %>% filter(!str_detect(word, "^[0-9]+([\\,]?[0-9]*)*$")) %>% anti_join(stop_words) %>%
    mutate(word = SnowballC::wordStem(word)))  # uses the Porter stemming algorithm to stem all the tokens to their root word
```

```
## # A tibble: 58,754 x 6
##       ID  cong billnum h_or_sen major word
##    <int> <int>   <int> <fct>    <int> <chr>
##  1     1   107    4499 HR          18 suspend
##  2     1   107    4499 HR          18 temporarili
##  3     1   107    4499 HR          18 duti
##  4     1   107    4499 HR          18 fast
##  5     1   107    4499 HR          18 magenta
##  6     1   107    4499 HR          18 stage
##  7     2   107    4500 HR          18 suspend
##  8     2   107    4500 HR          18 temporarili
##  9     2   107    4500 HR          18 duti
## 10     2   107    4500 HR          18 fast
## # ... with 58,744 more rows
```

Several packages implement stemming in R, including `hunspell` and `SnowballC`.

Most of these steps are to reduce the number of text features in the set of documents and thus **model complexity**.

Create Document-Term Matrices

Statistical learning algorithms require our data in a document-term matrix, i.e., a one-row-per-document format.

```
str(cast_dtm)
```

```
## function (data, document, term, value, weighting = tm::weightTf, ...)
```

```
congress_tokens %>% count(ID, word) %>% cast_dtm(document = ID, term = word, value = n)
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 4871)>>
## Non-/sparse entries: 54967/21616112
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency (tf)
```

Weighting

Term frequency (tf): how frequently a word occurs in a document.

- But it alone is not sufficiently helpful at teasing out important aspects and muting unimportant ones.

Inverse document frequency (idf): a measure that decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents.

Combining the two measures gives gives a term's **tf-idf**: the frequency of a term adjusted for how rarely it is used.

Suppose we have D documents total, a term's **tf-idf** is defined as:

$$(\text{\# of times } term_i \text{ appears}) \times \log\left(\frac{D}{\text{\# of documents with } term_i}\right)$$

`bind_tf_idf`

`tidytext`'s `bind_tf_idf` function takes a tidy text dataset as input with one row per token (term), per document.

```
str(bind_tf_idf)
```

```
## function (tbl, term, document, n)
```

```
congress_tokens %>% count(ID, word) %>% bind_tf_idf(word, ID, n)
```

```
## # A tibble: 54,967 x 6
##       ID word            n    tf   idf tf_idf
##    <int> <chr>       <int> <dbl> <dbl>  <dbl>
##  1     1 duti            1 0.167  2.65  0.441
##  2     1 fast            1 0.167  6.00  1.00
##  3     1 magenta         1 0.167  7.71  1.28
##  4     1 stage           1 0.167  5.57  0.928
##  5     1 suspend         1 0.167  2.94  0.490
##  6     1 temporarili     1 0.167  2.92  0.486
##  7     2 black           1 0.167  5.51  0.918
##  8     2 duti            1 0.167  2.65  0.441
##  9     2 fast            1 0.167  6.00  1.00
## 10     2 stage           1 0.167  5.57  0.928
## # ... with 54,957 more rows
```

To generate tf-idf for the document-term matrix, we can use the following workflow:

```
congress_tokens %>% count(ID, word) %>% bind_tf_idf(word, ID, n) %>% select(ID, word, tf_idf) %>% spread(word, tf_idf)
```

We can also change the weighting function in `cast_dtm()`:

```
(congress_dtm <- congress_tokens %>% count(ID, word) %>% cast_dtm(document = ID, term = word, value = n, weighting = tm::weightTfIdf))
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 4871)>>
## Non-/sparse entries: 54967/21616112
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

Sparsity

We can remove sparse terms from the model to further reduce model complexity using `removeSparseTerms` in the `tm` package:

```
tm::removeSparseTerms(congress_dtm, sparse = 0.9)
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 16)>>
## Non-/sparse entries: 14917/56267
## Sparsity           : 79%
## Maximal term length: 9
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

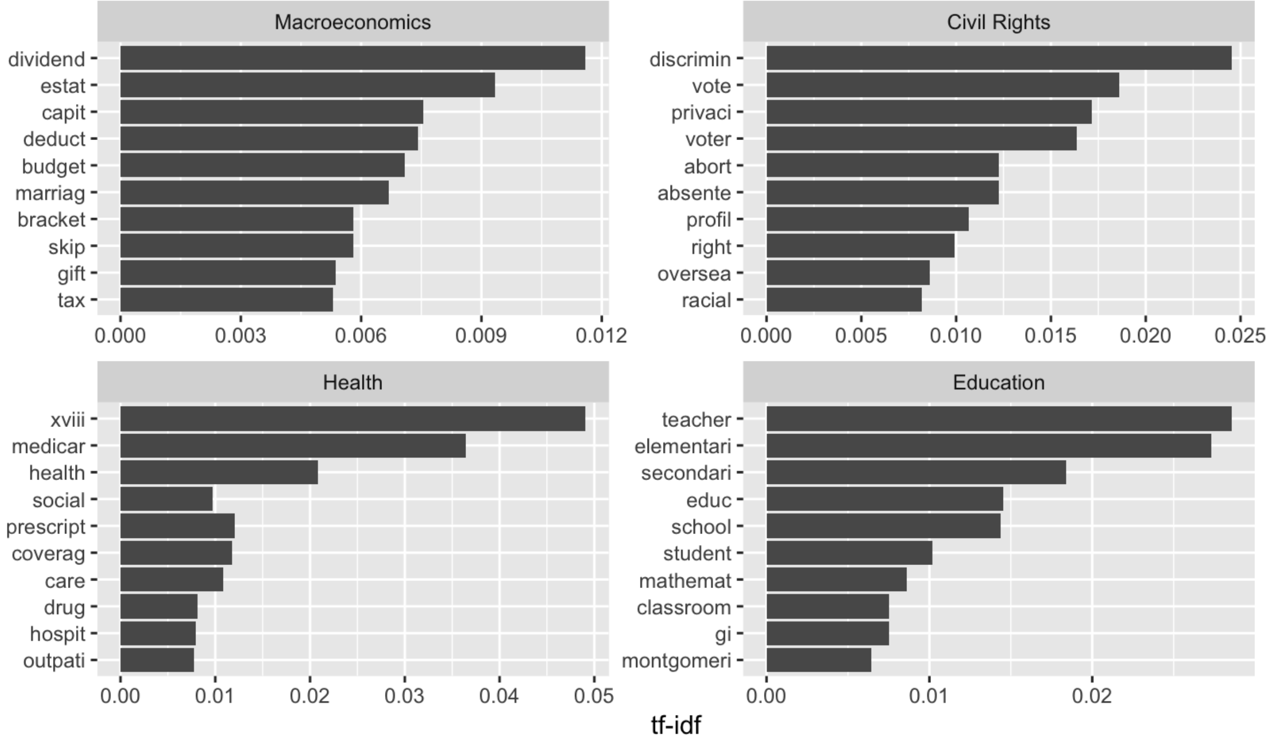
```
(congress_dtm <- tm::removeSparseTerms(congress_dtm, sparse = 0.99))
```

```
## <<DocumentTermMatrix (documents: 4449, terms: 209)>>
## Non-/sparse entries: 33794/896047
## Sparsity           : 96%
## Maximal term length: 11
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

Exploratory Analysis

```
plot_congress <- congress_tokens %>% count(major, word) %>% bind_tf_idf(term = word, document = major, n = n) %>% arrange(desc(tf_idf)) %>%
    mutate(word = factor(word, levels = rev(unique(word)))) %>% filter(major %in% c(1, 2, 3, 6)) %>% mutate(major = factor(major,
    levels = c(1, 2, 3, 6), labels = c("Macroeconomics", "Civil Rights", "Health", "Education"))) %>% group_by(major) %>%
    top_n(10) %>% ungroup()

plot_congress %>% ggplot(aes(word, tf_idf)) + geom_col() + labs(x = NULL, y = "tf-idf") + facet_wrap(~major, scales = "free") +
    coord_flip()
```



Estimating Statistical Models Using `caret`

`caret` is a package in R for training and plotting a wide variety of statistical learning models.

It does not contain the estimation algorithms itself; instead it creates a unified interface to hundreds of different models from various packages in R.

```
library(caret)
```

The basic function to train models is `train()`. Let's estimate a random forest:

```
congress_rf <- train(x = as.matrix(congress_dtm), y = factor(congress$major), method = "rf", ntree = 200, trControl = trainControl(method = "oob"))
```

```
congress_rf$finalModel
```

```
## 
## Call:
##  randomForest(x = x, y = y, ntree = 200, mtry = param$mtry) 
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 105
## 
##         OOB estimate of  error rate: 33.58%
## Confusion matrix:
##      1  2   3  4   5   6   7   8 10  12 13 14  15  16 17  18 19  20  21 99 class.error
## 1  107  0   3  0   5   3   2   2  3   6  3  2  15   2  0   1  1   7   1  0  0.34355828
## 2    0 16   1  0   5   3   6   1  1  11  5  2  10   4  2   2  0  11   4  0  0.80952381
## 3    7  1 526  4   8  10   3   0  3  13  8  2   9   9  1   3  1   7   2  0  0.14748784
## 4    2  0  12 83   3   2   6   0  1   3  0  1   6   0  0   4  1   3   5  1  0.37593985
## 5    7  0  16  2 142  10   7   1  3  18  2  2   8   5  4   8  3  13   9  2  0.45801527
## 6    4  0   6  1   9 164   3   1  1   8  1  1   5   1  3   6  3   2   3  0  0.26126126
## 7    6  1   6  7   2   3 108   5  8   7  0  1   5   4  1   1  3   6  27  0  0.46268657
## 8    7  0   1  1   0   0   7 102  1   1  0  1   5   0  0   2  1   3   6  0  0.26086957
## 10   9  1   1  2   5   1   2   0 88  20  0  0   3   3  2   4  1  19  10  0  0.48538012
## 12  10  4  21  2  14   5   7   0 12 144  2  4  12   7  2   5  5  29   4  2  0.50515464
## 13   7  0   4  1   5   3   1   2  1   4 55  1   0   1  4   0  0   3   2  0  0.41489362
## 14   2  0   2  1   4   1   5   2  3   3  2 40   4   2  3   2  1   2   1  0  0.50000000
## 15  17  4   7  9  10   2   6   2  5  19  1  1 156   4  4   8  6  12   6  0  0.44086022
## 16   2  1   3  0   6   2   3   1  5  10  1  2   6 138  0   4  7  15  13  0  0.36986301
## 17   4  1   5  1   7   3   1   1  2   7  0  3   5   2 36   1  1   7   3  0  0.60000000
## 18   1  0   0  4   6   1   4   2  1   2  0  0   2   0  1 368  5   4   1  0  0.08457711
## 19   2  0   3  3  11   4   7   0  2   6  0  1   5   5  0   6 52   6   8  0  0.57024793
## 20  11  2  10  0  14   5   8   1  5  21  2  2  14  11  3   9  2 246  13  1  0.35263158
## 21   7  2   6  1   4   5  28   3  4   7  1  4   5  10  1   4  1  18 361  0  0.23516949
## 99   0  0   0  0   3   0   0   0  1   0  0  0   0   1  1   0  0   0   1 23  0.23333333
```

```
randomForest::varImpPlot(congress_rf$finalModel)
```