Modeling the Bag of Words

POS6933: Computational Social Science

Jake S. Truscott, Ph.D

University of Florida
Spring 2026

Overview

- Housekeeping
- Week 5 Problem Set Review
- Dictionaries
- Multinomial Language Model
- Vector Space Model

Housekeeping

- Respond to comments re: topic selection by Friday at 5:00
- This week's material may require us to continue into next week – I will make determination by end of class.

Week 5 Problem Set Review

- Everyone generally did fine
- Make sure to fully read prompts...
- I don't mind y'all coordinating together – But more than half with same presidents, policies, and effectively the same vocabularies, it defeats a big chunk of the purpose.

Focus Today

- Focus today begins preliminary application of modeling strategies re: text as data
- Things to Consider:
  - Embrace the learning curve
  - Ask questions
  - If we need to continue this next week, we will!

Dictionaries

- A **dictionary** (*lexicon*) is a predefined list of words associated with categories (*classifications*)
- We can engage in basic classification or labeling tasks using these dictionaries to:
  1. Pre-processing our text to normalize/reduce complexity
  2. Counting how many dictionary words appear
  3. (Optional) Scaling document by length
  4. Assigning a category score based on the totals
- **Classification Tasks**: Assigning each document (or sentence) to one or more predefined categories based on its content – with dictionary methods, classification is completed by matching words.

Dictionaries for Classification Tasks – Binary Classification (e.g., *Positive* **or** *Negative*)

$$\text{Label}_i = \arg\max_c \left(\text{DictionaryMatches}_{ic}\right) \quad \textbf{or} \quad \text{Score} = \frac{\text{C1 Words} - \text{C2 Words}}{\text{Total Words}}$$

- i = Sentence, c = Label
- arg max = Label that best fits
- Score = Continuous Measure
- Example: 10 Positive Words and 12 Negative Words

$$10_{\text{Pos}} < 12_{\text{Neg}} \Rightarrow \text{Label} = \text{Neg}$$

$$-0.09 = \frac{10(\text{Pos}) - 12(\text{Neg})}{22(\text{Total})}$$

Dictionaries for Classification Tasks – Multiclass Classification (e.g., *Positive*, *Negative*, **or** *Neutral*)

$$\text{Label}_i = \underset{c \in \{\text{Pos},\text{Neg},\text{Neu}\}}{\arg\max} (\text{DictionaryMatches}_{ic}) \qquad \textbf{or} \qquad \text{Score} = \frac{\text{C}_c}{\text{Total Words}}$$

- *Note*: Continuous score now recovered as weight of each category in document.
- Example: 10 Positive Words, 12 Negative Words, 3 Neutral words

$$3_{\text{Neu}} < 10_{\text{Pos}} < 12_{\text{Neg}} \Rightarrow \text{Label} = \text{Neg}$$

$$(\text{Pos})\frac{10}{25} = 0.4 \quad (\text{Neg})\frac{12}{25} = 0.48 \quad (\text{Neu})\frac{3}{25} = 0.12$$

Dictionaries for Classification Tasks (Example)

- **Your Turn** – Work to recover classification labels for a binary task with 15 **positive** words and 11 **negative** words.
- Do the same for a multiclass task that now also includes 13 **neutral** words.

Dictionaries for Classification Tasks (Example Cont.)

- Binary:

$$15_{pos} > 11_{neg} \quad \text{or} \quad \frac{15-11}{26} \approx 0.15$$

- Multiclass:

$$(\text{Pos})\frac{15}{39} = 0.38 \quad (\text{Neg})\frac{11}{39} = 0.28 \quad (\text{Neu})\frac{13}{39} = 0.33$$

Thoughts re: Dictionaries

- Dictionary classifiers, while rather simple and intuitive, can nevertheless provide robust classification power.

Thoughts re: Dictionaries

- Dictionary classifiers, while rather simple and intuitive, can nevertheless provide robust classification power.
- **However** – much of that validity relies on the ability of researchers to construct dense lexicons (avoid small-N problems)

Thoughts re: Dictionaries

- Dictionary classifiers, while rather simple and intuitive, can nevertheless provide robust classification power.
- **However** – much of that validity relies on the ability of researchers to construct dense lexicons (avoid small-N problems)
- While a dense dictionary is certainly important, paramount to such is constructing categorical classifications (e.g., positive or negative) that are both authentic and defensible.

Thoughts re: Dictionaries

- Dictionary classifiers, while rather simple and intuitive, can nevertheless provide robust classification power.
- **However** – much of that validity relies on the ability of researchers to construct dense lexicons (avoid small-N problems)
- While a dense dictionary is certainly important, paramount to such is constructing categorical classifications (e.g., positive or negative) that are both authentic and defensible.
- Words have associated meaning based on tense and conditional usage. Ex: *happy* is certainly positive – but what if the sentence actually reads *I was not happy at all* (i.e., actually very negative!). Much of these concerns (as with other areas concerning the bag of words) can be improved given more robust methods, including the incorporation of inverses (not) and n-grams.

Creating Dictionaries

- Creating dictionaries is fairly intuitive – Create vectors (classes) with terms exclusively representing words unique to that classification.
- Ex:
  - **Positive**: Good, Great, Excellent, Benefit, Success
  - **Negative**: Bad, Poor, Failure, Harm, Risk
  - **Neutral**: Okay, Average, Fine, Moderate

Creating Dictionary in R

```
dictionary <- list(Positive = c("good", "great", "excellent", "benefit", "success"),
                   Negative = c("bad", "poor", "failure", "harm", "risk"),
                   Neutral  = c("okay", "average", "fine", "moderate")) # Dictionary as List

dictionary[['Positive']] # Sample
```

```
[1] "good"      "great"     "excellent" "benefit"   "success"
```

Applying Dictionary in R

```
sample_text <- reduce_complexity('The project had some success but also some risk') # Reduce Complexity
print(sample_text) # Print Sample
```

```
[1] "project success also risk"
```

```
sapply(dictionary, function(categories) sum(strsplit(sample_text, "\\W+")[[1]] %in% categories)) # Apply
```

```
Positive Negative  Neutral
       1        1        0
```

Applying Dictionary in R

- **Your Turn**: Create another string and apply the same dictionary.
- Afterwards – include additional values to the dictionary.

Existing Lexicons in R (BING)

- `BING` – lexicon widely used for binary sentiment classification.
- Assigns words to *positive* or *negative* classifications – approximately 2k positive words and 4.8k negative words

Existing Lexicons in R (BING – Cont.)

```
bing_dictionary <- tidytext::get_sentiments("bing") # Grab Dictionary

bing_dictionary %>%
  group_by(sentiment) %>%
  slice_sample(n = 3) %>%
  ungroup() %>%
  arrange(sentiment) # 3 Word Sample of Each
```

```
# A tibble: 6 x 2
  word        sentiment
  <chr>       <chr>
1 stupid      negative
2 dispiriting negative
3 nastiness   negative
4 engaging    positive
5 gained      positive
6 luxurious   positive
```

Existing Lexicons in R (BING – Cont.)

```
strings <- c("This decision is excellent, fair, and clearly the right outcome",
    "The opinion is good and persuasive, even if it is not perfect",
    "The ruling has some good points but also several serious flaws",
    "The decision is bad and poorly reasoned", "This opinion is terrible, deeply unfair, and completely w

strings <- sapply(strings, function(x) reduce_complexity(x),
    USE.NAMES = FALSE)
print(strings)
```

```
[1] "decision excellent fair clearly right outcome"   "opinion good persuasive even perfect"
[3] "rule good point also several serious flaw"       "decision bad poorly reason"
[5] "opinion terrible deeply unfair completely wrong"
```

Existing Lexicons in R (BING – Cont.)

```
strings <- tibble(
  doc_id = seq_along(strings),
  text = strings)

strings_tokens <- strings %>% # Convert to tibble
  tidytext::unnest_tokens(word, text) # Convert to Unnested Tokens

head(strings_tokens)
```

```
# A tibble: 6 x 2
  doc_id word
   <int> <chr>
1      1 decision
2      1 excellent
3      1 fair
4      1 clearly
5      1 right
6      1 outcome
```

Existing Lexicons in R (BING – Cont.)

```
strings_tokens %>%
  inner_join(tidytext::get_sentiments("bing"), by = "word") %>% # Get Sentiment
  group_by(doc_id, sentiment) %>%
  summarise(n = n(), # Total Word Matches
            words = paste(word, collapse = ", "), # Combine Word matches
            .groups = "drop") %>%
  left_join(strings, by = "doc_id") %>% # Add Back Original Text
  select(doc_id, text, sentiment, n, words) # Apply BING
```

```
# A tibble: 6 x 5
  doc_id text                                                sentiment     n words
   <int> <chr>                                               <chr>     <int> <chr>
1      1 decision excellent fair clearly right outcome       positive      4 excellent, fair, clearly, right
2      2 opinion good persuasive even perfect                positive      2 good, perfect
3      3 rule good point also several serious flaw           negative      1 flaw
4      3 rule good point also several serious flaw           positive      1 good
5      4 decision bad poorly reason                          negative      2 bad, poorly
6      5 opinion terrible deeply unfair completely wrong     negative      2 terrible, wrong
```

Existing Lexicons in R (AFINN)

- `AFINN` is another lexicon widely used in `R`
- Captures both direction **and** intensity of rhetoric
- Generally ranges from -5 (**Very Negative**) to +5 (**Very Positive**)

Existing Lexicons in R (AFINN – Cont.)

```
set.seed(123)
afinn_dictionary <- tidytext::get_sentiments("afinn") # Afinn Dictionary

afinn_dictionary %>%
  group_by(value) %>%
  slice_sample(n = 1) %>%
  ungroup() %>%
  arrange(value) %>%
  select(value, word) %>%
  { setNames(.$word, .$value) } # Sample Words (Value -5 to 5)
```

```
            -5             -4             -3             -2             -1              0
"son-of-a-bitch"  "fraudulence"    "lunatics"    "lethargy"  "manipulation"    "some kind"
             2              3              4              5
      "courtesy"     "cheery"      "winner"       "superb"
```

Existing Lexicons in R (AFINN – Cont.)

```
strings_tokens %>%
  inner_join(tidytext::get_sentiments(lexicon = 'afinn'), by = 'word') %>%
  group_by(doc_id, value) %>%
  summarise(n = n(), # Total Word Matches
            words = paste(word, collapse = ", "), # Combine Word matches
            .groups = "drop") %>%
  left_join(strings, by = "doc_id") %>% # Add Back Original Text
  select(doc_id, text, value, n, words)
```

```
# A tibble: 8 x 5
  doc_id text                                              value     n words
   <int> <chr>                                             <dbl> <int> <chr>
1      1 decision excellent fair clearly right outcome         1     1 clearly
2      1 decision excellent fair clearly right outcome         2     1 fair
3      1 decision excellent fair clearly right outcome         3     1 excellent
4      2 opinion good persuasive even perfect                  3     2 good, perfect
5      3 rule good point also several serious flaw             3     1 good
6      4 decision bad poorly reason                           -3     1 bad
7      5 opinion terrible deeply unfair completely wrong      -3     1 terrible
8      5 opinion terrible deeply unfair completely wrong      -2     2 unfair, wrong
```

Existing Lexicons in R (SentimentR)

- `sentimentR` uses a BING-style polarity lexicon (i.e., words have singular meaning) while also providing for **negators** (e.g., not, never, etc.), **amplifiers** (e.g., very or extremely), and **advesarial conjunction** (e.g., but).
- Result is a BING-style score combined with a valence multiplier to adjust for sentence heuristics (ex: *good* = 1x, not *good* = -1x, *barely good* = 0.5x)
- *Note*: Negative doesn't always flip the sign from positive to negative but it will help to scale the intensity.

Existing Lexicons in R (SentimentR - Cont.)

```
strings %>%
    mutate(sentiment = sentimentr::sentiment_by(text)$ave_sentiment) # Apply SentimentR
```

```
# A tibble: 5 x 3
  doc_id text                                          sentiment
   <int> <chr>                                             <dbl>
1      1 decision excellent fair clearly right outcome     1.14
2      2 opinion good persuasive even perfect              0.671
3      3 rule good point also several serious flaw        -0.0567
4      4 decision bad poorly reason                       -0.45
5      5 opinion terrible deeply unfair completely wrong  -1.18
```

Existing Lexicons in R (Example)

- **Your turn**: Recover scores from `Bing`, `AFINN`, and `SentimentR` after reducing the complexity of the following strings:
  - The recent peace agreement between the two nations is a remarkable step toward stability
  - The summit produced some promising proposals, though implementation will take time
  - The delegation met to discuss ongoing trade negotiations without reaching a conclusion
  - The sanctions imposed by the council are likely to harm the civilian population disproportionately
  - The military escalation is a disastrous and reckless move that threatens global security

Existing Lexicons in R (Example)

```
print(bing %>% select(text, doc_id))
```

```
# A tibble: 4 x 2
  text                                                                  doc_id
  <chr>                                                                  <int>
1 recent peace agreement two nation remarkable step toward stability         1
2 summit produce promise proposal though implementation will take time       2
3 sanction impose council likely harm civilian population disproportionately 4
4 military escalation disastrous reckless move threaten global security      5
```

```
print(bing %>% select(-c(text)))
```

```
# A tibble: 4 x 4
  doc_id sentiment     n words
   <int> <chr>     <int> <chr>
1      1 positive      3 peace, remarkable, stability
2      2 positive      1 promise
3      4 negative      2 impose, harm
4      5 negative      3 disastrous, reckless, threaten
```

Existing Lexicons in R (Example)

```
print(afinn %>% select(-c(text)))
```

```
# A tibble: 8 x 4
  doc_id value     n words
   <int> <dbl> <int> <chr>
1      1     1     1 agreement
2      1     2     2 peace, remarkable
3      2     1     1 promise
4      3     1     1 reach
5      4    -2     1 harm
6      4    -1     1 impose
7      5    -3     1 disastrous
8      5    -2     2 reckless, threaten
```

Existing Lexicons in R (Example)

```
print(sentimentr)
```

```
# A tibble: 5 x 3
  doc_id text                                                              sentiment
   <int> <chr>                                                                 <dbl>
1      1 recent peace agreement two nation remarkable step toward stability     0.833
2      2 summit produce promise proposal though implementation will take time   0.25
3      3 delegation meet discuss ongoing trade negotiation without reach conclusion    0
4      4 sanction impose council likely harm civilian population disproportionately   -0.389
5      5 military escalation disastrous reckless move threaten global security  -0.742
```

Multinomial Language Model

- **Multinomial Language Model**: A probabilistic model that treats a document as a bag of words generated from a multinomial distribution over a fixed vocabulary
- **Formally**: For a document represented as a sequence of word counts, the likelihood of observing the document is given by the multinomial probability mass function (PMF), which combines the factorial of the total word count with the product of the probabilities of each word raised to the power of its observed count
- Assumes that each word in a document is drawn independently from a fixed vocabulary according to a categorical distribution – i.e., in accordance with the Bag of Words approach, where each word has a certain probability of occurring.

Probability Mass Function – Categorical to Multinomial

PMF – Categorical Distribution:

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \prod_{j=1}^{J} \mu_j^{w_{ij}}$$

We can generalize for documents that are longer than one word using the multinomial distribution:

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{w}_{ij}}$$

Multinomail PMF (Explained)

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$$

- $p(\mathbf{W}_i \mid \boldsymbol{\mu})$ = probability of observing the entire word-count vector for document i given probabilities μ
- $\mathbf{M} = \sum_{j=1}^{j} \mathbf{W}_{ij}$ = Total number of word tokens in document i
- $\frac{M!}{\prod_{j=1}^{J} W_{ij}!}$ = Number of distinct word sequences consistent with the observed counts
- J = Vocabulary size (i.e., number of unique words)
- $\prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$ = Product of word probabilities raised to the number of times each word appears in document i

Multinomial Language Model – Food Example

- Vocabulary = `c(hamburger, salad, taco, nuggets)`
- Probabilities (μ):
  - p(hamburger) = 0.3
  - p(salad) = 0.25
  - p(taco) = 0.15
  - p(nuggets) = 0.3
- Count vector from Document i for `c(hamburger, salad, taco, nuggets)` = (2, 0, 1, 1)
- i.e., Hamburger (2), Salad (0), Taco (1), and Nuggets (1)

Multinomial Language Model – Food Example (Add Our Values – Simplify)

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$$

Multinomial Language Model – Food Example (Add Our Values – Simplify)

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$$

$$p(\texttt{H},\texttt{H},\texttt{T},\texttt{N} \mid \mu) = \frac{4!}{(2_H!)(0_S!)(1_T!)(1_N!)}(0.3_H)^2(0.25_S)^0(0.15_T)^1(0.3_N)^1$$

Multinomial Language Model – Food Example (Add Our Values – Simplify)

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$$

$$p(\mathtt{H},\mathtt{H},\mathtt{T},\mathtt{N} \mid \mu) = \frac{4!}{(2_H!)(0_S!)(1_T!)(1_N!)}(0.3_H)^2(0.25_S)^0(0.15_T)^1(0.3_N)^1$$

$$= \frac{4!}{2! \cdot 0! \cdot 1! \cdot 1!} \quad 0.09 \cdot 1 \cdot 0.15 \cdot 0.3$$

Multinomial Language Model – Food Example (Add Our Values – Simplify)

$$p(\mathbf{W}_i \mid \boldsymbol{\mu}) = \frac{M!}{\prod_{j=1}^{J} W_{ij}!} \prod_{j=1}^{J} \mu_j^{\mathbf{W}_{ij}}$$

$$p(\texttt{H},\texttt{H},\texttt{T},\texttt{N} \mid \mu) = \frac{4!}{(2_H!)(0_S!)(1_T!)(1_N!)}(0.3_H)^2(0.25_S)^0(0.15_T)^1(0.3_N)^1$$

$$= \frac{4!}{2! \cdot 0! \cdot 1! \cdot 1!} \quad 0.09 \cdot 1 \cdot 0.15 \cdot 0.3$$

$$p(H, H, T, N \mid \mu) \approx 0.0486$$

Federalist Papers

- Collection of essays published in NY newspapers advocating ratification of US Constitution
- Published anonymously by James Madison, Alexander Hamilton, and Jon Jay – all using pseudonym *Publius*
- Virtually any course on American politics prescribes Federalist 10 (*republics and factionalism*) and 51 (*separation of powers to prevent tyranny*) – I also prescribe 78 (*Judiciary*)

Mosteller and Wallace (1963) – Cont.

- 85 essays in total – some where authorship was known, others not – and some disputed.
- By mid-20th century, it was believed that Jay authored 5, Hamilton (at least) 43, and Madison (at least) 14
- Left several with disputed authorship.
- Mosteller and Wallace (1963) used Bag of Words assumption to try and prescribe unknown authorship.
- **Basic Idea**: Variance in each potential author's word choice should emerge in disputed essay.

Prescribing Authorship for Federalist 51

What We Need:

Prescribing Authorship for Federalist 51

What We Need:

- A vocabulary

Prescribing Authorship for Federalist 51

What We Need:

- A vocabulary

```
by, heretofore, man, upon, whilst
```

Prescribing Authorship for Federalist 51

What We Need:

- A vocabulary

```
by, heretofore, man, upon, whilst
```

- Variance of that vocabulary in a document of interest *i*

Prescribing Authorship for Federalist 51

What We Need:

- A vocabulary

  `by, heretofore, man, upon, whilst`

- Variance of that vocabulary in a document of interest i
- The associated probabilities μ

Authorship of Federalist 51 (Cont.)

```
tidy_federalist %>%
  count(author, word) %>%
  tidyr:: pivot_wider(names_from = author,values_from = n, values_fill = 0) %>%
  {
    wide <- .
    bind_rows(
      wide,
      wide %>%
        select(-word) %>%
        summarise(across(everything(), sum)) %>%
        mutate(word = "TOTAL") %>%
        select(word, everything()))
  }
```

Authorship of Federalist 51 (Vocab Frequencies by Author)

```
# A tibble: 6 x 4
  word       Hamilton   Jay Madison
  <chr>         <int> <int>   <int>
1 by              861    82     477
2 heretofore       13     1       1
3 man             102     0      17
4 upon            374     1       7
5 whilst            1     0      12
6 TOTAL          1351    84     514
```

Authorship of Federalist 51 (Recovering μ)

$$W_{Hamilton} \text{Multinomial}(1351, \mu_H)$$

$$W_{Madison} \text{Multinomial}(514, \mu_M)$$

$$W_{Jay} \text{Multinomial}(84, \mu_J)$$

Authorship of Federalist 51 (Recovering $\mu_{H,M,J}$) – Cont.

$$\mu_{\sigma j} = \frac{W_{\sigma j}}{N_\sigma}$$

Hamilton:

$$\mu_{Hamilton} = (\frac{861}{861+13+102+374+1}, \frac{13}{1351}, \frac{102}{1351}, \frac{374}{1351}, \frac{1}{1351})$$

Hamilton:

$$\mu_{Hamilton} = (0.63, 0.009, 0.07, 0.27, 0.0007)$$

Authorship of Federalist 51 (Recovering $\mu_{H,M,J}$) – Cont.

Solve for Madison

Authorship of Federalist 51 (Recovering $\mu_{H,M,J}$) – Cont.

Solve for Madison

Madison:

$$\mu_{Madison} = (0.92, 0.001, 0.033, 0.013, 0.023)$$

Jay:

$$\mu_{Jay} = (0.97, 0.01, 0, 0.01, 0)$$

Authorship of Federalist 51 – Vocabulary in 51

- We know the following frequency of the vocabulary in Federalist 51:
  - by (23)
  - man (1)
  - upon (0)
  - heretofore (0)
  - whilst (2)

- **Next Step**: Plug in our values!

Federalist 51 – Putting Together: Hamilton

$$p(\mathbf{W}_{\mathbf{Fed51}} \mid \mu_{Hamilton}) = \frac{26!}{(23!)(1!)(0!)(0!)(2!)}(0.63)^{23}(0.009)^{1}(0.07)^{0}(0.27)^{0}(0.0007)^{2}$$

Federalist 51 – Putting Together: Hamilton

$$p(\mathbf{W_{Fed51}} \mid \mu_{Hamilton}) = \frac{26!}{(23!)(1!)(0!)(0!)(2!)}(0.63)^{23}(0.009)^{1}(0.07)^{0}(0.27)^{0}(0.0007)^{2}$$

$$p(\mathbf{W_{Fed51}} \mid \mu_{Hamilton}) = 0.0000000008346$$

Federalist 51 – Putting Together: Madison

- **Your Turn** – Do the same for Madison

Federalist 51 – Putting Together: Madison

- **Your Turn** – Do the same for Madison

$$p(\mathbf{W_{Fed51}} \mid \mu_{Madison}) = \frac{26!}{(23!)(1!)(0!)(0!)(2!)}(0.92)^{23}(0.001)^{1}(0.033)^{0}(0.013)^{0}(0.023)^{2}$$

$$p(\mathbf{W_{Fed51}} \mid \mu_{Madison}) = 0.00055692$$

Federalist 51 – Putting Together: Jay

$$p(\mathbf{W_{Fed51}} \mid \mu_{Jay}) = \frac{26!}{(23!)(1!)(0!)(0!)(2!)}(0.97)^{23}(0.01)^{1}(0)^{0}(0.01)^{0}(0)^{2}$$

$$p(\mathbf{W_{Fed51}} \mid \mu_{Jay}) = 0$$

Federalist 51 (Comparison)

$$p(\mathbf{W}_{\mathbf{Fed51}} \mid \mu_{Madison}) > p(\mathbf{W}_{\mathbf{Fed51}} \mid \mu_{Hamilton}) > p(\mathbf{W}_{\mathbf{Fed51}} \mid \mu_{Jay})$$

- Supports findings by Mosteller and Wallace (1963)

Multinomial Language Model – Laplace Smoothing

- Other consideration – no usage of `man` or `whilst` by John Jay – produces a probability of 0.
- Alternative – **Laplace Smoothing**: Add a positive integer to all the word vector values to remove the impossibility while otherwise preserving the proportionality of word usage

Multinomial Language Model – Laplace Smoothing (Cont.)

```
hamilton_likelihood <- dmultinom(x = federalist_51_vector, prob = author_vectors[['Hamilton']] + 1)

madison_likelihood <- dmultinom(x = federalist_51_vector, prob = author_vectors[['Madison']] + 1)

jay_likelihood <- dmultinom(x = federalist_51_vector, prob = author_vectors[['Jay']] + 1)


data.frame(Author = c('Hamilton', 'Madison', 'Jay'),
           Likelihood = c(hamilton_likelihood, madison_likelihood, jay_likelihood))
```

```
    Author   Likelihood
1 Hamilton 3.845094e-08
2  Madison 2.557079e-02
3      Jay 2.222033e-03
```

```
madison_likelihood/jay_likelihood # Likelihood Ratio of Madison v. Jay
```

```
[1] 11.50783
```

```
madison_likelihood/hamilton_likelihood # Likelihood Ratio of Madison v. Hamilton
```

```
[1] 665023.7
```

Multinomial Language Model – Dirichlet Distribution

- Another alternative – rather than adding a numeric constant to the word counts, add a **Dirichlet prior** over μ, allowing the word probabilities themselves to be treated as random and regularized through pseudo-count parameters (α).
- Basically: When calculating $\mu_{\sigma j}$, add a prior weight value (α_j) that reflects our prior belief about how common that word is expected to be before observing the document.
- For example, I am going to assign $\alpha = (2, 1, 1, 2, 1)$

Multinomial Language Model – Updated $\mu_H \sim \alpha = (2, 1, 1, 2, 1)$

$$\hat{\mu}_{Hamilton} = (\frac{861 + 2}{861 + 13 + 102 + 374 + 1 + (2 + 1 + 1 + 2 + 1)}, \frac{13 + 1}{1358}, \frac{102 + 1}{1358}, \frac{374 + 2}{1358}, \frac{1 + 1}{1358})$$

$$p(\mathbf{W}_{\mathbf{Fed51}} \mid \widehat{\mu}_{\mathbf{Hamilton}}) = \frac{26!}{(23!)(1!)(0!)(0!)(2!)}(0.63)^{23}(0.01)^{1}(0.07)^{0}(0.27)^{0}(0.001)^{2}$$

$$p(\mathbf{W}_{\mathbf{Fed51}} \mid \widehat{\mu}_{\mathbf{Hamilton}}) = 0.00000000186$$

Vector Space Model

- While multinomial language model looks at text as a series of words and focuses on how likely each word is to appear, a vector space model treats text as a point in space.
- Lets us measure how similar two texts are based on distance or direction.
- In essence, **MLM** is all about word probabilities – figuring out which words are more likely in a document
- **VSMs** leverage linear algebra to turn text into vectors so we can compare documents based on their overall content, not just exact word counts.

Vector Space Model – Ex: Inner Products (Federalist 51)

$$\mathbf{W}_{\text{Madison}} \cdot \mathbf{W}_{\text{Hamilton}} = (477, 1, 17, 7, 12) \cdot (861, 13, 102, 374, 1)$$

$$= (477 \times 861) + (1 \times 13) + (17 \times 102) + (7 \times 374) + (12 \times 1)$$

$$= 410,697$$

$$\mathbf{W}_{\text{Madison}} \cdot \mathbf{W}_{\text{Jay}} = (477, 1, 17, 7, 12) \cdot (82, 1, 0, 1, 0, 84)$$

$$= (477 \times 82) + (1 \times 1) + (17 \times 0) + (7 \times 0) + (12 \times 84)$$

$$= 40,123$$

Cosine Similarity

- Recall the distribution of frequencies across Hamilton, Madison, and Jay seemed to heavily favor Hamilton
- **Important Q**: Should it matter more that Hamilton used the four of the five words in our vocabulary more frequently than Madison, or should it matter more how the distribution of that word usage matches that in the disputed document?
- Ideally, no – but high-dimensionality makes this very possible.
- **Cosine Similarity** allows us to normalize the inner product and the magnitude of the vectors.

Cosine Similarity (Cont.)

$$\text{cosine(u,v)} = \frac{\text{u} \cdot \text{v}}{||\text{u}|| \ ||\text{v}||} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2}\sqrt{\sum_i v_i^2}}$$

- Numerator = Inner product of the two vectors
- Denominator = Product of the vectors' magnitudes

Cosine Similarity – Hamilton & Federalist 51 Example

- **Inner Product**:

$$\mathbf{W}_{Hamilton} \cdot \mathbf{W}_{Disputed} = (861 \times 23) + (13 \times 1) + (102 \times 0) + (374 \times 0) + (1 \times 2) \quad = 19,818$$

Cosine Similarity – Hamilton & Federalist 51 Example

- **Inner Product**:

$$\mathbf{W}_{Hamilton} \cdot \mathbf{W}_{Disputed} = (861 \times 23) + (13 \times 1) + (102 \times 0) + (374 \times 0) + (1 \times 2) \quad = 19,818$$

- **Magnitude of Vectors (Hamilton)**:

$$||\mathbf{W}_{Hamilton}|| = \sqrt{861^2 + 13^2 + 102^2 + 374^2 + 1^2} \quad \approx 944.33$$

Cosine Similarity – Hamilton & Federalist 51 Example

- **Inner Product**:

$$\mathbf{W}_{Hamilton} \cdot \mathbf{W}_{Disputed} = (861 \times 23) + (13 \times 1) + (102 \times 0) + (374 \times 0) + (1 \times 2) \quad = 19,818$$

- **Magnitude of Vectors (Hamilton)**:

$$||\mathbf{W}_{Hamilton}|| = \sqrt{861^2 + 13^2 + 102^2 + 374^2 + 1^2} \quad \approx 944.33$$

- **Magnitude of Vectors** (Federalist 51):

$$||\mathbf{W}_{Disputed}|| = \sqrt{23^2 + 1^2 + 0^2 + 0^2 + 2^2} \quad \approx 23.10$$

Cosine Similarity – Hamilton & Federalist 51 Example

- **Inner Product**:

$$\mathbf{W}_{Hamilton} \cdot \mathbf{W}_{Disputed} = (861 \times 23) + (13 \times 1) + (102 \times 0) + (374 \times 0) + (1 \times 2) \quad = 19,818$$

- **Magnitude of Vectors (Hamilton)**:

$$||\mathbf{W}_{Hamilton}|| = \sqrt{861^2 + 13^2 + 102^2 + 374^2 + 1^2} \quad \approx 944.33$$

- **Magnitude of Vectors** (Federalist 51):

$$||\mathbf{W}_{Disputed}|| = \sqrt{23^2 + 1^2 + 0^2 + 0^2 + 2^2} \quad \approx 23.10$$

- **Cosine Similarity**:

$$cos(\mathbf{W}_{Hamilton}, \mathbf{W}_{Disputed}) = \frac{19,818}{944.33 \times 23.10} \quad \approx 0.9085$$

Cosine Similarity – Madison

- **Your turn** – Try with Madison

Cosine Similarity – Madison

- **Your turn** – Try with Madison

$$cos(\mathbf{W}_{Madison}, \mathbf{W}_{Disputed}) = \frac{10,996}{477.50 \times 23.10} \approx 0.996$$

Cosine Similarity – Madison

- **Your turn** – Try with Madison

$$cos(\mathbf{W}_{Madison}, \mathbf{W}_{Disputed}) = \frac{10,996}{477.50 \times 23.10} \approx 0.996$$

- Effectively removes all doubt re: Madison's authorship!

TF-IDF

- **Term Frequency - Inverse Frequency** (TF-IDF): Strategy that rescales a DFM by its inverse document frequency, down-weighting terms that appear in many documents.
- In Short: Words that occur frequently in individual documents while remaining relatively uncommon in the broader corpus receive greater weight
- Terms that satisfy both conditions are in the *Goldilocks zone* – those that violate both are penalized.

TF-IDF (Cont.)

$$W_{ij}^{\text{tf-idf}} = W_{ij} * log\frac{N}{n_j}$$

$N =$ Number of Documents in Corpus

$n_j =$ Number of Documents Containing Word j

$W_{ij} =$ Word Count

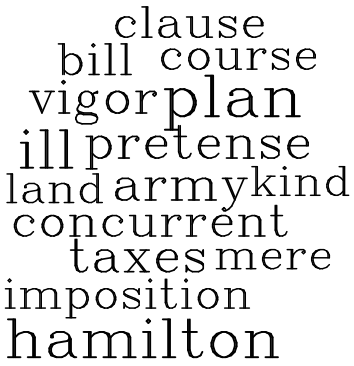
$log\frac{N}{n_j} =$ Penalty for Frequent Words

Hamilton – No TF-IDF

```
wordcloud::wordcloud(words = top_hamilton$word,
                     freq = top_hamilton$n,
                     vfont=c("serif","plain")) # No TF-IDF
```

Hamilton – TF-IDF

```
wordcloud::wordcloud(words = top_hamilton$word,
                     freq = top_hamilton$tf_idf,
                     vfont=c("serif","plain")) # TF-IDF
```



Next Class

- Next Class: Supervised Learning Methods (Naive Bayes)
- **Remember**: Work on final projects!