Topic 3: Modelling Texts and Discriminating Words

LSE MY459: Computational Text Analysis and Large Language Models

https://lse-my459.github.io/

Ryan Hübert
Associate Professor of Methodology

Slides last updated: 2 February 2026

Modelling texts

Last week: practical approach to quantifying texts

- ➜ Quantify by counting token frequency
- ➜ Used “bag of words”: word order doesn’t matter
- ➜ Some issues around feature selection (to reduce size of DFM)

Once we turn text into a DFM, we’ve made texts *comparable*

- ➜ Comparison is the life-blood of social science!

Next question: how do we formalise comparisons?

- ➜ We’ll refer to this as **modelling texts**

There are two fundamental ways to model texts:

1. **Vector space approach**: points in a space
2. **Probabilistic approach**: outputs of a process

Texts as points in a space

Vector space approach

Vector space approach: each document is mathematically represented as a **vector** in a J-dimensional **vector space**

- ➜ Each document vector is $\mathbf{W}_i$ from the DFM $\mathbf{W}$

Useful because:

- ➜ Vectors are *geometric* representations
- ➜ Can be characterised in terms of points in space where there are (precise) mathematical notions of "closeness"

Keep in mind:

- ➜ Here, we're focused on comparisons across documents
- ➜ Vector representations are also useful for characterising *features* within documents, e.g. word embeddings

Vector representation for a simple corpus

Consider a simple example of three documents:

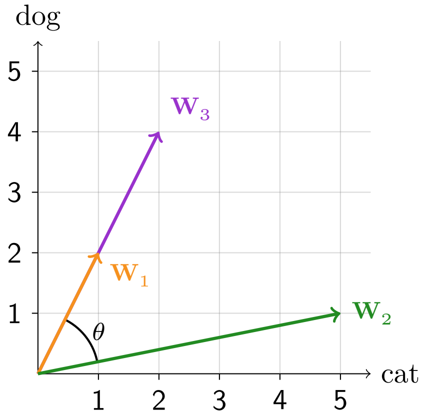
- Document 1: `cat dog dog`
- Document 2: `dog cat cat cat cat cat`
- Document 3: `dog dog cat dog cat dog`

The document feature matrix would look like:

```
             cat   dog
Document 1     1     2
Document 2     5     1
Document 3     2     4
```

So: $\mathbf{W}_1 = (1, 2)$, $\mathbf{W}_2 = (5, 1)$ and $\mathbf{W}_3 = (2, 4)$

Vector representation for a simple corpus

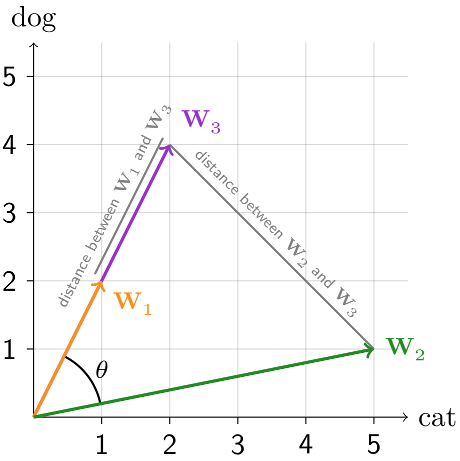


Core question: how do we *compare* these documents?

1. Distance: absolute separation between points
2. Similarity: directional closeness of vectors

Distance: absolute separation between points

Examples of distance between documents:


dog
cat
5
4
3
2
1
1
2
3
4
5
$\mathbf{w}_3$
$\mathbf{w}_1$
$\mathbf{w}_2$
θ
distance between $\mathbf{w}_1$ and $\mathbf{w}_3$
distance between $\mathbf{w}_2$ and $\mathbf{w}_3$

Euclidean distance

The previous slide illustrated the **Euclidean distance** between documents $\mathbf{W}_1$ and $\mathbf{W}_3$, and between $\mathbf{W}_2$ and $\mathbf{W}_3$

➜ Euclidean distance is based on the Pythagorean theorem

For two documents $\mathbf{W}_A$ and $\mathbf{W}_B$ with J features:

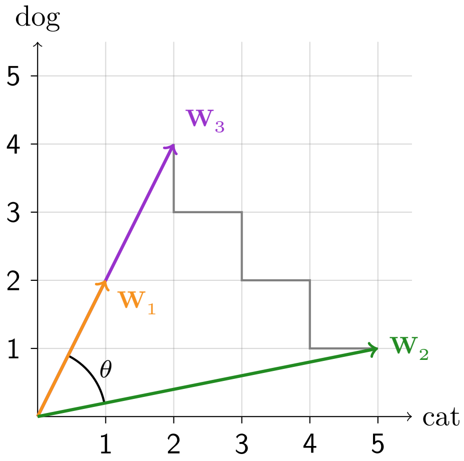
$$d_E(\mathbf{W}_A, \mathbf{W}_B) = \|\mathbf{W}_A - \mathbf{W}_B\| = \sqrt{\sum_{j=1}^{J} (W_{Aj} - W_{Bj})^2}$$

So, for $\mathbf{W}_2 = (5, 1)$ and $\mathbf{W}_3 = (2, 4)$:

$$\|\mathbf{W}_2 - \mathbf{W}_3\| = \sqrt{(5-2)^2 + (1-4)^2} \approx 4.24$$

Other distance metrics

The **Manhattan distance** (or **taxicab distance**) is calculated assuming you can only travel in right angles



Other distance metrics

It is calculated with this formula:

$$d_T(\mathbf{W}_A, \mathbf{W}_B) = \sum_j |W_{Aj} - W_{Bj}|$$

So, for $\mathbf{W}_2 = (5, 1)$ and $\mathbf{W}_3 = (2, 4)$:

$$d_T(\mathbf{W}_2, \mathbf{W}_3) = |5 - 2| + |1 - 4| = 6$$

There are other distance metrics (e.g., Minkowski and Mahalanobis)

➜ Choice of distance metrics is up to the analyst based on the task

Similarity: directional closeness

Document similarity: how *directionally close* are documents?

It's not obvious how to measure closeness, example from GRS:

- ➜ Are a positive review and a negative review of the same film "closer" than two positive reviews of different films?

Four desirable properties for document similarity measures:

1. Maximum similarity: comparing document to itself
2. Minimum similarity: two documents with no words in common
   - ➜ These are **orthogonal** documents
3. Similarity increases as more of same words used
4. Symmetry: similarity of $\mathbf{W}_A$ to $\mathbf{W}_B$ is the same as similarity of $\mathbf{W}_B$ to $\mathbf{W}_A$

The inner product

One straight-forward similarity metric: **inner product**

For two documents $\mathbf{W}_A$ and $\mathbf{W}_B$ with J features:

$$\mathbf{W}_A \cdot \mathbf{W}_B = W_{A1}W_{B1} + W_{A2}W_{B2} + \cdots + W_{AJ}W_{BJ}$$

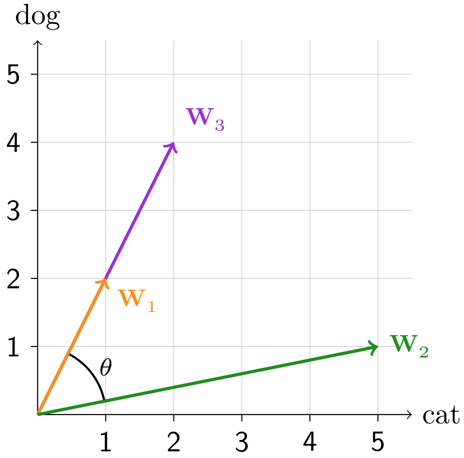
So, for $\mathbf{W}_2 = (5, 1)$ and $\mathbf{W}_3 = (2, 4)$:

$$\mathbf{W}_2 \cdot \mathbf{W}_3 = 5 \times 2 + 1 \times 4 = 14$$

Major downside: inner product is very sensitive to vector magnitude

- ➜ Vector magnitude is the length of vector—NOT same as document length
- ➜ Problem because similarity is about *directional* closeness

The inner product



$\mathbf{w}_1 \cdot \mathbf{w}_2 = 1 \times 5 + 2 \times 1 = 7$ $\quad$ $\mathbf{w}_2 \cdot \mathbf{w}_3 = 5 \times 2 + 1 \times 4 = 14$

Vector magnitude

To deal with this, let's *normalise* each document vector by its **vector magnitude** before calculating the inner product

- ➜ Vector magnitude is just the Euclidean distance between the vector and the origin, notated as $\|\mathbf{x}\|$ for a vector $\mathbf{x}$
- ➜ (In 2D, this is the Pythagorean theorem!)

The magnitude of a vector $\mathbf{x}$ of length J can be calculated

$$\|\mathbf{x}\| = \sqrt{(x_1 - 0)^2 + \cdots + (x_J - 0)^2} = \sqrt{x_1^2 + \cdots + x_J^2}$$

A document $\mathbf{W}_A$ can be normalised by dividing by its magnitude:

$$\frac{\mathbf{W}_A}{\|\mathbf{W}_A\|} = \frac{\mathbf{W}_A}{\sqrt{\sum_{j=1}^{J} W_{Aj}^2}}$$

Cosine similarity

The **cosine similarity** between two documents $\mathbf{W}_A$ and $\mathbf{W}_B$ is just the inner product of the normalised document vectors:

$$\text{cosine similarity}(\mathbf{W}_A, \mathbf{W}_B) = \frac{\mathbf{W}_A}{\|\mathbf{W}_A\|} \cdot \frac{\mathbf{W}_B}{\|\mathbf{W}_B\|}$$

Why is it called “cosine similarity”?

$$\text{cosine similarity}(\mathbf{W}_A, \mathbf{W}_B) = \frac{\mathbf{W}_A}{\|\mathbf{W}_A\|} \cdot \frac{\mathbf{W}_B}{\|\mathbf{W}_B\|} = \cos\theta$$

Cosine similarity is based on the size of the angle between vectors—their *directional* closeness!

Metric ranges from from 0 to 1, where 0 is least similar (“orthogonal”, 90° angles) and 1 is most similar

Cosine similarity

So, for $\mathbf{W}_2 = (5, 1)$ and $\mathbf{W}_3 = (2, 4)$:

$$\text{cosine similarity}(\mathbf{W}_2, \mathbf{W}_3) = \frac{5 \times 2 + 1 \times 4}{\sqrt{(5^2 + 1^2)\,(2^2 + 4^2)}} \approx 0.614$$

And if we want to know the angle between these documents

$$\cos\theta = 0.614 \iff \theta = \arccos(0.614) \approx 0.9098 \text{ radians}$$

(This is approximately 52°)

There are other similarity metrics, but we won't cover them

Texts as outputs of processes

Limits of vector space approach

Vector space approach: excellent for measuring similarity of documents

But it treats documents as finished objects:

1. No way of thinking about *uncertainty*
   - ➜ Documents have fundamental and sampling uncertainty
   - ➜ Similarity metrics confound *true* differences and noise
2. No model of how text is produced
   - ➜ Vectors summarise what happened, not why
   - ➜ There's no possibility for generation or forecasting

Vector space approach tells us how texts differ, but **probabilistic approach** tells us whether differences are meaningful

Probabilistic approach

Basic idea: documents are "draws" from a probability distribution

Think back to your stats courses:

- A **data generating process** is a probability distribution that (we assume) captures the way real-life data occurs
- Real-life datasets are conceptualised as **samples** from a DGP
- In some sense, the DGP is what we care about since it tells us "how things work"
- We use real-life data to try to learn about the DGP, which we never observe
- If we do a "good job" learning about the DGP, we can generate new texts (this is what LLMs do)

Probabilistic models

Under this approach, there are many possible assumptions one can make about the DGP

- ➜ These are different **(probabilistic) models**

For example, consider a DFM:

- ➜ It is an *empirical* distribution
- ➜ Under the probabilistic approach, it represents a sample from some DGP
- ➜ Each draw is a vector of word counts
- ➜ As a sample, it inherits sampling properties

But, what kinds of DGPs would generate this kind of empirical distribution?

Multinomial model

One of the simplest probabilistic models that generates a DFM: assume each document $\mathbf{W}_i$ is a draw from a **multinomial distribution** with two parameters:

- $\boldsymbol{\mu} \rightarrow$ a J-length vector of probabilities that each feature in the J-feature vocabulary is drawn
- $M_i \rightarrow$ the document length (number of tokens)

In math notation, $\mathbf{W}_i \sim \text{Multinomial}(M_i, \boldsymbol{\mu})$

Diagram for 3-document model

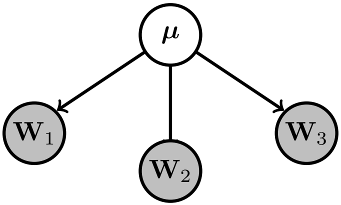
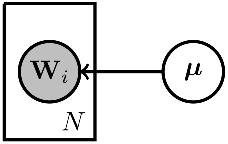


Diagram for N-document model (plate diagram)



Very simple example

Suppose a vocabulary with $J = 3$ features: `cat`, `dog`, `rat`

Further suppose that `cat` is twice as likely as `dog` or `rat`, which are equally likely:

$$\boldsymbol{\mu} = (\mu_{\text{cat}} = 0.50, \mu_{\text{dog}} = 0.25, \mu_{\text{rat}} = 0.25)$$

We've made an *assumption* about the way documents are made (i.e., the DGP)

Very simple example

Suppose our assumption is correct, and then we can *simulate* drawing ("generating") four documents of different lengths

To do this in Python, we'll use tools available in the `numpy` and `scipy` modules

```
import numpy as np
from scipy.stats import multinomial
```

First, define some objects

```
mu = [0.5, 0.25, 0.25]           # assumed feature probabilities
M = [2, 3, 2, 7]                 # document lengths
rng = np.random.default_rng(873) # seed for reproducibility
```

Very simple example

Here we generate four “documents” from our model

```python
for Mi in M:
    multinomial.rvs(n=Mi, p=mu, random_state=rng)
```

```
array([2, 0, 0])
array([0, 2, 1])
array([1, 1, 0])
array([2, 1, 4])
```

Generated “documents” are vectors of token counts,
e.g. $\mathbf{W}_4 = (2, 1, 4)$

Can be stacked into a DFM:

```
             cat   dog   rat
Document 1     2     0     0
Document 2     0     2     1
Document 3     1     1     0
Document 4     2     1     4
```

Calculating important stuff

There are other DGPs that could generate the kind of empirical distributions that we see in DFMs

It is common to use the multinomial distribution as the DGP for a DFM because it's easy to work with

➜ Can look up various important formulas

For example, what if we want to calculate the probability of getting a particular document $\mathbf{W}_i$, given $\boldsymbol{\mu}$:

$$p(\mathbf{W}_i|\boldsymbol{\mu}) = \frac{M_i!}{\prod_{j=1}^{J}\left(W_{ij}!\right)}\prod_{j=1}^{J}\left(\mu_j^{W_{ij}}\right)$$

Calculating important stuff

Back to our simple example with three word vocabulary:

$$\boldsymbol{\mu} = (\mu_{\text{cat}} = 0.50, \mu_{\text{dog}} = 0.25, \mu_{\text{rat}} = 0.25)$$

What is probability of getting a document $\mathbf{W}_i = (2, 1, 1)$?

$$p(\mathbf{W}_i|\boldsymbol{\mu}) = \frac{(2+1+1)!}{2! \times 1! \times 1!}\left[0.5^2 \times 0.25^1 \times 0.25^1\right] = \frac{12}{64} = 0.1875$$

This is the probability of getting a "bag of words" with 2 cats, 1 dog, and 1 rat, e.g.: `cat rat dog cat`, or `rat dog cat cat`, or `dog rat cat cat`, etc.

- ➜ Word order doesn't matter (once we represent as counts)
- ➜ So, this DGP matches the bag of words approach

Very simple example

We can calculate in Python as well

```
mu = np.array([0.5, 0.25, 0.25])
Wi = np.array([2, 1, 1])
p = multinomial.pmf(Wi, n=Wi.sum(), p=mu)
print(round(p,6))
```

```
0.1875
```

Estimating models with data

In real-world situations, we don't usually know the DGP

- E.g., we don't know $\boldsymbol{\mu}$ and need to estimate it from data

How? Due to properties of the multinomial distribution, it's simple

1. If all documents are drawn from their own model:

$$\hat{\boldsymbol{\mu}}_i = (\hat{\mu}_{i1}, ..., \hat{\mu}_{iJ}) = \left(\frac{W_{i1}}{M_i}, ..., \frac{W_{iJ}}{M_i} \right)$$

2. If all documents are drawn according to $\boldsymbol{\mu}$:

$$\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, ..., \hat{\mu}_J) = \left(\frac{\sum_i W_{i1}}{\sum_i M_i}, ..., \frac{\sum_i W_{iJ}}{\sum_i M_i} \right)$$

Very useful for social science: can have *category-specific* models

Example: the Federalist papers

When the US was established, three people wrote a set of essays called The Federalist Papers in support of the new US Constitution

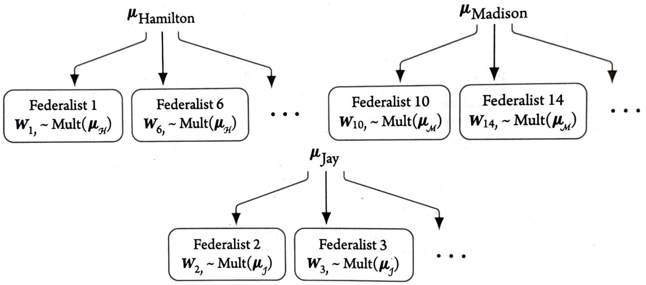
- The people: Alexander Hamilton, John Jay, James Madison
- There were 85, none of them had bylines (all "Publius")
- Historical records tell us who wrote 73 of them; remainder are contested and thus remain "unlabelled"

Mosteller and Wallace (1963) use a probability model to try to figure out who wrote the unlabelled Federalist papers

Example: the Federalist papers

Consider three category-specific probabilistic models, where each category k is an author (and $K = 3$ is number of categories)

- ➜ Substantively, captures author's distinctive "writing style"
- ➜ Formally, each writer's $\boldsymbol{\mu}$ is different: $\boldsymbol{\mu}_{\mathcal{H}}$, $\boldsymbol{\mu}_{\mathcal{J}}$, $\boldsymbol{\mu}_{\mathcal{M}}$



μ_{Hamilton}
Federalist 1
$W_1 \sim \text{Mult}(\mu_{\mathcal{H}})$
Federalist 6
$W_6 \sim \text{Mult}(\mu_{\mathcal{H}})$
...
μ_{Madison}
Federalist 10
$W_{10} \sim \text{Mult}(\mu_{\mathcal{M}})$
Federalist 14
$W_{14} \sim \text{Mult}(\mu_{\mathcal{M}})$
...
μ_{Jay}
Federalist 2
$W_2 \sim \text{Mult}(\mu_{\mathcal{J}})$
Federalist 3
$W_3 \sim \text{Mult}(\mu_{\mathcal{J}})$
...

Source: Figure 6.3 of GRS (p. 64)

Example: the Federalist papers

The basic process:

Step 1: estimate $\boldsymbol{\mu}$ for each writer ("figure out their writing style") using real-world data, generating estimates: $\hat{\boldsymbol{\mu}}_{\mathcal{H}}$, $\hat{\boldsymbol{\mu}}_{\mathcal{J}}$, $\hat{\boldsymbol{\mu}}_{\mathcal{M}}$

Step 2: see which model best explains the unlabelled documents by calculating probability each person was the author of the unlabelled documents

Since we assume each author's style is constant across documents, we can act like each author wrote one big document:

- ➜ Sum token counts across each author's document rows
- ➜ Works due to properties of multinomial distribution

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabelled ones?

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabelled ones?

Step 1: estimate Hamilton's "writing style" $\hat{\boldsymbol{\mu}}_{\mathcal{H}}$

$$\hat{\boldsymbol{\mu}}_{\mathcal{H}} = \left(\frac{859}{859+102+374}, \frac{102}{859+102+374}, \frac{374}{859+102+374} \right) \approx (0.64, 0.08, 0.28)$$

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabelled ones?

Step 1: estimate Hamilton's "writing style" $\hat{\boldsymbol{\mu}}_{\mathcal{H}}$

$$\hat{\boldsymbol{\mu}}_{\mathcal{H}} = \left(\frac{859}{859+102+374}, \frac{102}{859+102+374}, \frac{374}{859+102+374} \right) \approx (0.64, 0.08, 0.28)$$

Step 2: calculate probability Hamilton wrote unlabeled documents

$$\Pr(\mathbf{W}_{\text{Unlabeled}} | \hat{\boldsymbol{\mu}}_{\mathcal{H}}) = \frac{(15+2+0)!}{15! \times 2! \times 0!} \times 0.64^{15} \times 0.08^{2} \times 0.28^{0} = 0.001$$

Conclude: approx. 0.1% chance Hamilton wrote disputed documents

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

Can repeat for Madison:

$$\hat{\boldsymbol{\mu}}_{\mathcal{M}} = \left(\frac{474}{474+17+7}, \frac{17}{474+17+7}, \frac{7}{474+17+7}\right) \approx (0.95, 0.035, 0.015)$$

$$\Pr(\mathbf{W}_{\textsf{Unlabeled}} | \hat{\boldsymbol{\mu}}_{\mathcal{M}}) = \frac{(15+2+0)!}{15! \times 2! \times 0!} \times 0.95^{15} \times 0.035^{2} \times 0.015^{0} = 0.077$$

Conclude: approx. 8% chance Madison wrote disputed documents

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

And for Jay:

$$\hat{\boldsymbol{\mu}}_{\mathcal{J}} = \left(\frac{82}{82+0+1}, \frac{0}{82+0+1}, \frac{1}{82+0+1} \right) \approx (0.99, 0, 0.01)$$

$$\Pr(\mathbf{W}_{\text{Unlabeled}} | \hat{\boldsymbol{\mu}}_{\mathcal{J}}) = \frac{(15+2+0)!}{15! \times 2! \times 0!} \times 0.99^{15} \times 0^{2} \times 0.01^{0} = 0$$

Conclude: 0% chance Jay wrote disputed documents

Overfitting due to low word counts

We conclude Madison is most likely to be author of unlabelled texts

But, there's a weird issue: we predicted a 0% chance Jay wrote the documents

- This is an example of **overfitting**: lessons drawn from data (a sample) are "too" closely tied to the patterns in the data

Problem: Jay never used the word "man" in labelled documents

- Estimated model: any document with "man" can never be written by Jay
- Our model of Jay's word use is overfitting the sample data
- No use of "man" is likely just a sampling fluke—surely Jay knew and used the word "man"!

General issue with sparse DFMs

Regularisation and smoothing

We solve this with **regularisation**: "a non-data piece of information or a constraint that is added to the model to draw estimates toward a particular value." (GSR, p. 66)

Many approaches to regularisation, but a very easy one in this context is **Laplace smoothing**

➜ Add a little bit of fake data (a constant α) to ensure no zeroes

We can add $\alpha = 1$ to our *labelled* Federalist papers data

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

	by	man	upon
Hamilton	860	103	375
Jay	83	1	2
Madison	475	18	8
Unlabeled	15	2	0

Regularisation and smoothing

We solve this with **regularisation**: "a non-data piece of information or a constraint that is added to the model to draw estimates toward a particular value." (GSR, p. 66)

Many approaches to regularisation, but a very easy one in this context is **Laplace smoothing**

➜ Add a little bit of fake data (a constant α) to ensure no zeroes

For example we can add $\alpha = 1$ to our Federalist papers data

Changes calculations slightly, e.g. for a category-level model:

$$\hat{\mu}_{kj} = \frac{W_{kj} + \alpha}{M_k + \sum_j \alpha_j}$$

Regularisation and smoothing

Let's recalculate Federalist probabilities with this regularised data

	by	man	upon
Hamilton	860	103	375
Jay	83	1	2
Madison	475	18	8
Unlabeled	15	2	0

For example, probability the unlabeled texts are written by Jay

$$\hat{\boldsymbol{\mu}}_{\mathcal{J}} = \left(\frac{83}{83+1+2}, \frac{1}{83+1+2}, \frac{2}{83+1+2}\right) \approx (0.97, 0.01, 0.02)$$

$$\Pr(\mathbf{W}_{\textsf{Unlabeled}}|\hat{\boldsymbol{\mu}}_{\mathcal{J}}) = \frac{(15+2+0)!}{15! \times 2! \times 0!} \times 0.97^{15} \times 0.01^{2} \times 0.02^{0} = 0.0086$$

Connecting this with more general models

So far, we just looked at a *specific* kind of DGP that could generate data in the *specific* form of a DFM

More generally, a **language model** (or LM) is a probability distribution over sequences of tokens

Formally, for any sequence of M tokens, an LM specifies:

$$\Pr(t_1, t_2, ..., t_M) \equiv \Pr(t_{1:M})$$

Using the chain rule of probability:

$$\Pr(t_{1:M}) = \Pr(t_1) \cdot \Pr(t_2|t_1) \cdot \Pr(t_3|t_1, t_2) \cdots \Pr(t_M|t_1, t_2, ..., t_{M-1})$$

An LM provides a way to *generate* tokens in sequence

Estimation challenges

Remember: we don't see DGPs, so for a LM to be useful for generation, you need to estimate it

To estimate $\Pr(t_{1:M})$, you need to be able to estimate $\Pr(t_m | t_{1:(m-1)})$ for all $m \leq M$

These are very difficult to estimate in real life settings

- ➜ Language is creative and datasets are sparse
- ➜ There is *some* non-zero probability that someone will say "Mars" after saying "My name is Ryan and I am from"
- ➜ In real life datasets, there are probably no instances of this

Core problem: a general LM is often too complex

Markovian assumption

When a model is too complex, simplify with assumptions

For a LM, we can use a **Markovian** assumption: only need to condition on a smaller number of prior words

A language model with a Markovian assumption is known as an n-gram language model

- The Markovian assumption: only $n-1$ prior words affect the probability of the next word

For example, for the mth token in a sequence:

- Bigram language model only requires: $\Pr(t_m|t_{m-1})$
- Trigram language model only requires: $\Pr(t_m|t_{m-2}, t_{m-1})$
- Unigram language model only requires: $\Pr(t_m)$

Connecting to the multinomial model

The multinomial model of language gave us a data generating process for “whole documents”

- ➜ Formally: vectors of token counts

The multinomial model could not accommodate word order: it’s too simple

So, a DFM created by ignoring word order (bag of words) is an appropriate representation of a sample drawn from this simple DGP

How does it connect to the unigram language model?

Connecting to the multinomial model

Suppose we have a document with the following sequence of tokens $(t_1, t_2, t_3, ..., t_M)$, and a vocabulary $V = (v_1, v_2, ..., v_J)$

➜ Note: In the next few slides I suppress document index i

Under the unigram language model:

$$\Pr(t_{1:M}) = \Pr(t_1) \cdot \Pr(t_2) \cdots \Pr(t_M)$$

Let W_j be the count of feature j in the document, then this is equivalent to:

$$\Pr(t_{1:M}) = \Pr(v_1)^{W_1} \Pr(v_2)^{W_2} \cdots \Pr(v_J)^{W_J}$$

Connecting to the multinomial model

Under the multinomial model, recall that $\mu_j = \Pr(v_j)$ for all j

$$\Pr(t_{1:M}) = \mu_1^{W_1} \mu_2^{W_2} \cdots \mu_J^{W_J}$$

This gives us a probability of the *specific* sequence $\Pr(t_{1:M})$

Remember that the multinomial model returns probabilities over token counts: $\Pr(W_1, W_2, W_3, ..., W_J)$

- Many sequences would give the exact same word counts
- In fact there are $M!/(W_1! \cdots W_J!)$ sequences that yield these same word counts as sequence $t_{1:M}$

$$\Pr(W_1, ..., W_J) = \frac{M!}{W_1! \cdots W_J!} \Pr(t_{1:M}) = \frac{M!}{\prod_{j=1}^{J} W_j!} \prod_{j=1}^{J} \mu_j^{W_j}$$

Language models today

Again: an LM is a probability distribution over token sequences

Traditionally, there were lots of estimation challenges

- Models with Markovian assumptions only preserve short-range word order
- So, LMs with strong Markovian assumptions struggle to generate intelligible sequences of tokens
- Fine for lots of social science, not fine for chatbots

Technical architecture underlying large language models “solved” this problem

- More detail later in course
- Basic idea: we can now estimate conditional probabilities on very large sequences of words

Discriminating words

Discriminating words

Vector space and probabilistic approaches give us a structured way to represent documents using their word usage

- ➜ Documents that use language similarly end up close together
- ➜ We'll see more of this in future weeks!

Often, similar documents correspond to known categories, e.g.:

- ➜ rhetorical differences between Democrats and Republicans (Monroe, Colaresi and Quinn 2008)
- ➜ rhetorical differences between first-wave and second-wave feminist movements in the US (Nelson 2020)

Once categories are known, a natural question is: which words are driving the separation between them?

We call these **discriminating words** (or "key words")

Detecting discriminating words

Suppose you have N documents and each document i fits into one of K known categories, e.g.,

- ➜ partisanship of author, e.g. Democrat v. Republican
- ➜ time period of document, e.g. first wave v. second wave feminist movements

We'll use k as a generic placeholder for a specific category

The core question: *for each feature (e.g. "word")* j in a corpus's vocabulary, how much does the presence of that feature in a document inform our guess about the document's category?

- ➜ Process: go feature-by-feature and calculate how much that feature discriminates

We'll consider simple examples with two categories

Running example

A corpus of inaugural addresses given by U.S. presidents

- ➜ Contains every address from each presidential inauguration (1789 to 2025)
- ➜ Sourced from the `{quanteda}` R package (see here)
- ➜ In R, can access directly as `data_corpus_inaugural` after `library("quanteda")`
- ➜ Available as a `.csv` file in the `data` repo of the course GitHub

Our question for now: which words distinguish Trump's 2017 address, relative to all other post-war presidents' addresses?

Contingency table

A useful starting point is a **contingency table**

This is a general concept from statistics, but in our case, it will look like a “consolidated” DFM

Specifically:

- ➜ consolidate all features that are not j into a feature “not j”
- ➜ consolidate all documents into their categories

Here: “consolidate” means add together word counts

Contingency table

Let's measure how discriminating the word "America" is between Trump's 2017 address and all other postwar presidents

The contingency table looks like this:

	america	not america
not Trump	167	17490
Trump	19	694

Some useful quantities from the table:

- ➜ Total tokens: $W = 167 + 19 + 17490 + 694 = 18370$
- ➜ Total instances of "america": $W_{\mathsf{a}} = 167 + 19 = 186$
- ➜ Total tokens by Trump: $W_{\mathcal{T}} = 19 + 694 = 713$
- ➜ Total instances of "america" by Trump: $W_{\mathcal{T}\mathsf{a}} = 19$

Two measurement approaches

The contingency table gives us the basic word counts we need to do our calculations

Once you have it, then, what's next?

Two approaches for measuring feature discrimination among known categories:

1. Statistical association measures
2. Model-based approaches ("fightin' words")

Statistical association measures

We will use the logic of hypothesis testing from statistics

1. Set up an expectation based on an **independence assumption** that there is no correlation between features and categories

   ➜ In math, assume that $\Pr(j,k) = \Pr(j)\Pr(k)$ for every j and k

2. Estimate $\widehat{\Pr}_{H_0}(j,k)$ using data, such as:

$$\widehat{\Pr}_{H_0}(j,k) = \widehat{\Pr}(\texttt{america})\widehat{\Pr}(\textsf{Trump}) = \frac{186}{18370} \times \frac{713}{18370} = 0.0003930$$

3. Calculate expected word counts under independence assumption, given corpus size, such as:

$$E_{\mathcal{T}_{\texttt{a}}} = 18370 \times 0.0003930 \approx 7.21941$$

Hypothetical contingency table

We can repeat for each combination of feature and category, and create a *hypothetical* contingency table assuming independence

In a 18,370-token corpus, here's what we would expect to see if Trump was no more or less likely to use the token "america"

	america	not america
not Trump	178.8	17478.2
Trump	7.2	705.8

Keep in mind: we will notate word counts in the real contingency table using W, and word counts in the hypothetical contingency table using E

Test statistics

These two contingency tables allow us to calculate two different test statistics:

1. The **likelihood ratio statistic** (G^2):

$$G^2 = 2 \times \sum_k \sum_j \left(W_{kj} \times \log\left(\frac{W_{kj}}{E_{kj}}\right)\right)$$

2. The **Pearson's** χ^2 statistic:

$$\chi^2 = \sum_k \sum_j \left(\frac{\left(W_{kj} - E_{kj}\right)^2}{E_{kj}} \right)$$

(These two statistics converge in large samples)

Statistical association measures

Both G^2 and χ^2 are test statistics for the same null hypothesis (H_0): author and token use are independent

➜ More generally: the variable on the rows and columns of the contingency table are independent

Under H_0, each test statistic is distributed according to the χ^2 distribution with one parameter, the degrees of freedom (df)

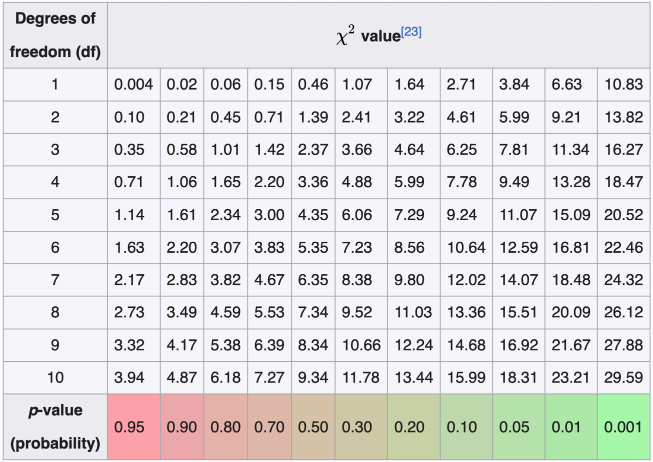
➜ df is calculated as: $(K-1) \times (J-1)$

We reject H_0 if the test statistic is sufficiently large

➜ How far? Depends on the critical level of significance α we choose, usually 0.05

Statistical association measures

We can use a standard table of p-values, like:



Degrees of freedom (df)	χ^2 value[23]										
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.63	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.61	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.81	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.87	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59
p-value (probability)	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001

Source: Wikipedia

Statistical association measures

Let's calculate G^2 for the token "america"

Recall the formula:

$$G^2 = 2 \times \sum_k \sum_j \left(W_{kj} \times \log \left(\frac{W_{kj}}{E_{kj}} \right) \right)$$

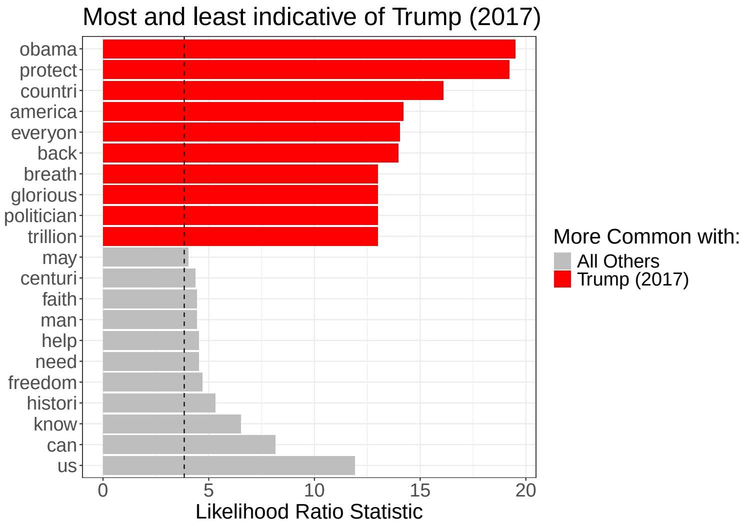
Recall our two tables:

	america	not america
not Trump	167	17490
Trump	19	694

	america	not america
not Trump	178.8	17478.2
Trump	7.2	705.8

$$G^2 = 2 \times \Big[\left(167 \times \log\left(\frac{167}{178.8}\right)\right) + \left(19 \times \log\left(\frac{19}{7.2}\right)\right) + \left(17490 \times \log\left(\frac{17490}{17478.2}\right)\right) + \left(694 \times \log\left(\frac{694}{705.8}\right)\right) \Big] \approx 14.3$$

Statistical association measures



Discriminating words with a language model

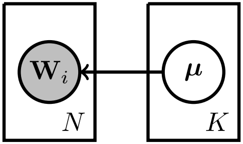
Statistical association measures: posit independence model, then calculate likelihood the observed text is consistent with it

A different way to identify discriminating words uses a probabilistic model of language:

- Model Trump's language directly and estimate how word use differs
- This is known as the **fightin' words** approach to measuring discriminating words
- Developed in Monroe, Colaresi and Quinn (2008)

Language model

Let's assume a language model like the following:



Specifically:

- $K = 2$, so $\boldsymbol{\mu} = [\boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\mu}_{\mathcal{O}}]$ ($\mathcal{O} =$ other postwar presidents)
- $J = 2$, so each of $\boldsymbol{\mu}_{\mathcal{T}}$ and $\boldsymbol{\mu}_{\mathcal{O}}$ has a length of 2

Again, because DFMs are sparse, we probably want to regularise

- And we will use Laplace smoothing with $\alpha = 1$

(Note: original paper regularises differently)

Estimating $\boldsymbol{\mu}$ with data

We can estimate using this formula

$$\hat{\mu}_{kj} = \frac{W_{kj} + \alpha_j}{W_k + \sum_j \alpha_j}$$

For each category k (Trump or Other):

$$\widehat{\boldsymbol{\mu}}_{\mathcal{T}} = \left[\hat{\mu}_{\mathcal{T}j}, 1 - \hat{\mu}_{\mathcal{T}j}\right] \qquad \widehat{\boldsymbol{\mu}}_{\mathcal{O}} = \left[\hat{\mu}_{\mathcal{O}j}, 1 - \hat{\mu}_{\mathcal{O}j}\right]$$

Estimating $\boldsymbol{\mu}$ with data

Let's do this for "america" — recall the contingency table

	america	not america
not Trump	167	17490
Trump	19	694

So:

$$\widehat{\mu}_{\mathcal{T}_\mathsf{a}} = \frac{19+1}{19+694+1+1} = 0.0280 \quad \widehat{\mu}_{\mathcal{O}_\mathsf{a}} = \frac{167+1}{167+17490+1+1} = 0.0095$$

We can see right away: our estimates tell us Trump has a higher probability of using "america" than others

➜ How to use them to calculate a standard "score" for the word?

Measuring word discrimination

Monroe, Colaresi and Quinn (2008) recommend log-odds difference for quantifying how discriminating a word j is between group k and group k':

$$\hat{\delta}_j^{(k-k')} = \log\left(\frac{\widehat{\mu}_{kj}}{1-\widehat{\mu}_{kj}}\right) - \log\left(\frac{\widehat{\mu}_{k'j}}{1-\widehat{\mu}_{k'j}}\right)$$

This puts too much weight on infrequent words, so we can normalise to a z-score as follows:

$$\hat{z}_j^{(k-k')} = \frac{\hat{\delta}_j^{(k-k')}}{\sqrt{\mathrm{Var}\left(\hat{\delta}_j^{(k-k')}\right)}}$$

Where:

$$\mathrm{Var}\left(\hat{\delta}_j^{(k-k')}\right) \approx \frac{1}{W_{kj}+\alpha_j} + \frac{1}{W_{k'j}+\alpha_j}$$

Measuring word discrimination

Back to our example, we can use $\hat{\mu}_{\mathcal{T}\text{a}}, \hat{\mu}_{\mathcal{O}\text{a}}$ to calculate:

$$\hat{\delta}_{\text{a}}^{(\mathcal{T}-\mathcal{O})} = \log\left(\frac{0.0280}{0.972}\right) - \log\left(\frac{0.0095}{0.9905}\right) \approx 1.10$$

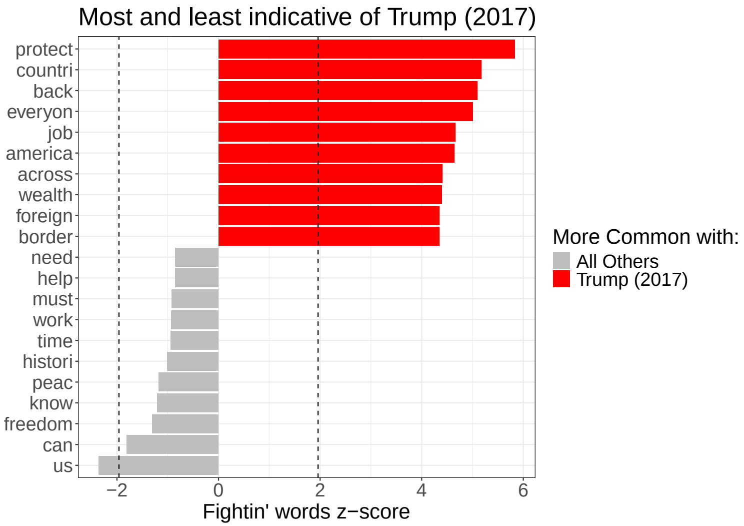
and

$$\operatorname{Var}\left(\hat{\delta}_{\text{a}}^{(\mathcal{T}-\mathcal{O})}\right) = \frac{1}{20} + \frac{1}{168} \approx 0.0560$$

This yields a standardised z-score of

$$\hat{z}_j^{(\mathcal{T}-\mathcal{O})} \approx \frac{1.10}{\sqrt{0.0560}} = 4.65$$

Measuring word discrimination


Most and least indicative of Trump (2017)
protect
countri
back
everyon
job
america
across
wealth
foreign
border
need
help
must
work
time
histori
peac
know
freedom
can
us
Fightin' words z−score
More Common with:
All Others
Trump (2017)