Topic 5: Word Counting Methods

LSE MY459: Computational Text Analysis and Large Language Models

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

Ryan Hübert
Associate Professor of Methodology

Slides last updated: 15 February 2026

From discovery to measurement

Last topic: methods to *discover* latent structure in a corpus

- ➜ Specifically, ways to organise documents into groups
- ➜ We had very little information on the latent structure
- ➜ Lots of validation required

Now: move toward measuring **theory-defined concepts** in texts

- ➜ Dictionary methods and supervised scaling: measure concepts in documents by systematically counting and weighting word usage relative to dictionaries or reference texts (today)
- ➜ Classification: use labelled data to assign documents to predefined categories (next topic)

Automated dictionary methods

Word meanings

An obvious point: words have meanings!

So, word use in a document can give us a sense of the overall "meaning" of that document

What kind of by "meanings"? Some examples:

- Sentiment: positive or negative, etc.
- Emotions: sad, happy, angry, anxious, etc.
- Hate speech: sexism, homophobia, xenophobia, racism, etc.

Word meanings usually map to concepts of interest to social scientists

Word meanings

Automated dictionary methods (ADMs) exploit word meanings to learn about concepts in documents

Two important steps:

1. Construct list of words and associated concepts called a **dictionary**
2. Quantify concepts in documents based on word counts and mapping between dictionary words and concepts

Important: dictionary has to be appropriate for your task, and requires **validation** (more later)

Structure of a dictionary

A **dictionary** is a pre-defined list of document features (e.g., words) associated with specific concepts

Two important components of dictionaries:

1. **Keys**: labels for the concepts of interest
2. **Values**: terms or patterns that “match” the concepts defined by the keys

Conceptually similar to `dict` objects in Python

An example

A simple example:

```
import re
my_dict = {"christmas": {"Christmas", "Santa", "holiday"},
           "opposition": {"Opposition", "reject"},
           "taxregex": re.compile(r"tax.+$"),
           "country": {"United States", "Sweden"}}
```

(Example inspired by https://quanteda.io/reference/dfm_lookup.html)

Several keys matched to sets of words or regex patterns

Demonstrates both conceptual flexibility and “look up” flexibility

- Looking for Christmas-related and tax-related words
- Looking for exact matches of specific words like “holiday” and also regex patterns like `tax.+$`

(Might want to preprocess this dictionary, depending on task!)

A real-world dictionary

LexiCoder Sentiment Dictionary (LSD): for sentiment analysis in political texts, validated with human-coded news content

Available at https://www.snsoroka.com/data-lexicoder

- Download: https://www.snsoroka.com/s/LSDaug2015.zip
- User agreement: https://www.snsoroka.com/s/LSDagreement.pdf
- Citation: Young, L. and Soroka, S. 2012. Lexicoder Sentiment Dictionary.

Contains four lists of globs:

- positive
- negative
- neg_positive (count as "negative")
- neg_negative (count as "positive")

A real-world dictionary

For example, the file `LSD2015.lc3`:

```
+negative#AA0000
a lie
abandon*
abas*
abattoir*
abdicat*
aberra*
abhor*
abject*
abnormal*
abolish*
abominab*
abominat*
abrasiv*
...
```

```
+positive#008800
ability*
abound*
absolv*
absorbent*
absorption*
abundanc*
abundant*
acced*
accentuat*
accept*
accessib*
acclaim*
acclamation*
...
```

A real-world dictionary

For example, the file `LSD2015_NEG.lc3`:

```
+neg_positive
 best not
 better not
 no damag*
 no no
 not ability*
 not able
 not abound*
 not absolv*
 not absorbent*
 not absorption*
 not abundanc*
 not abundant*
 not acced*
...
```

```
+neg_negative
 not a lie
 not abandon*
 not abas*
 not abattoir*
 not abdicat*
 not aberra*
 not abhor*
 not abject*
 not abnormal*
 not abolish*
 not abominab*
 not abominat*
 not abrasiv*
...
```

Calculating quantities of interest

Once you have a dictionary, you can quantify the concepts in the dictionary by counting word use

Specifically, for a concept k, an ADM assumes a vector $\boldsymbol{\mu}_k = (\mu_{k1}, ..., \mu_{kJ})$ that specifies how to weight each feature j

- ➜ How much does that feature contribute to the measure of the concept?

Simple sentiment analysis example:

- ➜ One concept k: “sentiment” or “tone” (can drop k index)
- ➜ For each j in the set of “positive” words, set $\mu_j = 1$
- ➜ For each j in the set of “negative” words, set $\mu_j = -1$
- ➜ For all other features j, set $\mu_j = 0$

Calculating quantities of interest

With vectors in hand, you can calculate a **document score** for each document i:

$$\hat{\pi}_i = \sum_{j=1}^{J} \mu_j W_{ij}$$

For example: a document i's overall "sentiment"/"tone"

Depending on application, you can normalise by document length:

$$\hat{\pi}_i^{\mathsf{n}} = \frac{1}{M_i} \hat{\pi}_i = \frac{1}{M_i} \sum_{j=1}^{J} \mu_j W_{ij}$$

Or convert to a binary classification, e.g.:

$$\hat{\pi}_i^{\mathsf{cl}} = \begin{cases} 1 & \text{if } \hat{\pi}_i \geq 0 \\ -1 & \text{if } \hat{\pi}_i < 0 \end{cases}$$

Calculating quantities of interest: example

If you have a dictionary, you can set up a simple Python pipeline to calculate document scores

Using LSD, some basic "cleaning" steps yield four regex patterns to find phrases in the four categories (more in seminar)

```
pos_rx
```

```
re.compile("(?:^|\\s+)(merit\\w*|sanguin\\w*|okay|impartial\\w*|coping\\w*|a
```

```
neg_rx
```

```
re.compile("(?:^|\\s+)(abstrus\\w*|crucify\\w*|deluded\\w*|mundan\\w*|scares
```

```
negn_rx
```

```
re.compile("(?:^|\\s+)(not\\ rampant|not\\ ambiguit\\w*|not\\ burn|not\\ ghas
```

```
negp_rx
```

```
re.compile("(?:^|\\s+)(not\\ elaborat\\w*|not\\ accept\\w*|not\\ ebullient\\w
```

Calculating quantities of interest: example

Consider a simple text:

```
text = "She abandoned me and it's not great!"
text
```

```
"She abandoned me and it's not great!"
```

Preprocess it:

```
text = text.lower()
text = re.sub(r"[^\w \'\-]+", "", text)
text
```

```
"she abandoned me and it's not great"
```

How long is it (for normalising)?

```
tlen = len(re.split(r"\s+", text))
tlen
```

```
7
```

Calculating quantities of interest: example

Figure out which words go into each category

```
# Detect negations
s = {"pos" : [], "neg" : [], "neg_pos": [], "neg_neg": []}
s["neg_neg"] = re.findall(negn_rx, text)
s["neg_pos"] = re.findall(negp_rx, text)

# Drop detected negations from text (no double counting!)
for k in s["neg_pos"] + s["neg_neg"]:
    text = text.replace(k, "")

# Detect remaining
s["neg"] = re.findall(neg_rx, text)
s["pos"] = re.findall(pos_rx, text)

s
```

```
{'pos': [], 'neg': ['abandoned'], 'neg_pos': ['not great'], 'neg_neg': []}
```

Calculating quantities of interest: example

Now, calculate the document score

```
doc_score = len(s["pos"]) + len(s["neg_neg"])      # pos. phrases
doc_score += - len(s["neg"]) - len(s["neg_pos"])  # neg. phrases
doc_score = doc_score / tlen                       # normalise
doc_score
```

```
-0.2857142857142857
```

Can calculate by hand:

$$\hat{\pi}_i = \frac{0 + 0 - 1 - 1}{7} \approx -0.286$$

There are lots of dictionaries...

- **General Inquirer**: an early "all purpose" dictionary
- **Regressive Imagery Dictionary**: designed to measure primordial vs. conceptual thinking
- **Linguistic Inquiry and Word Count** or **LIWC** (pronounced "Luke"): large dictionary + (paid) software to generate analyses
- **SentiStrength**: For social media text
- **Moral Foundations Dictionary**: meant to capture "moral foundations" concepts

Always: read the docs, use correctly and for intended use cases!

A dictionary in nltk

Valence Aware Dictionary and sEntiment Reasoner or (**VADER**): open source, and designed to work well for modern social media text

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sentence = "I absolutely LOVE MY459!"
sid = SentimentIntensityAnalyzer()
sid.polarity_scores(sentence)
```

```
{'neg': 0.0, 'neu': 0.266, 'pos': 0.734, 'compound': 0.7592}
```

The `compound` score is an "overall" score:

- Greater than 0.05 $\rightarrow$ positive
- Less than -0.05 $\rightarrow$ negative

A dictionary in `nltk`

VADER uses a more complicated weighting scheme

- ➜ Words are given different weight sizes based on *how* positive and *how* negative they are (not just 1 and -1)
- ➜ Weights are based on human coding
- ➜ Also, picks up on other cues: capitalisation and punctuation

```
sentence2 = "I absolutely love MY459."
sid.polarity_scores(sentence2)
```

```
{'neg': 0.0, 'neu': 0.308, 'pos': 0.692, 'compound': 0.6697}
```

VADER's document scores are (roughly):

$$\widehat{\pi}_i^{\text{VADER}} = \sum_{j=1}^{J} \mu_j \lambda_{ij} W_{ij}$$

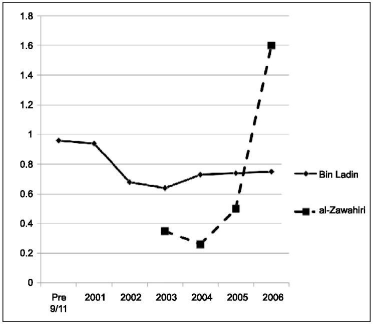
where λ_{ij} scales for "local context"

Example: Terrorist speech

Pennebaker and Chung (2008) and Hancock et al. (2010)

- ➜ Use ADMs to analyse Al Qaeda discourse in videotapes, interviews and letters

From Hancock et al. (2010):



- ➜ Key Finding: Zawahiri was feeling threatened, indicating a rift with bin Laden
- ➜ Use of first-person pronouns (I, me, my, mine): bin Laden's use remained constant, Zawahri's use increased

Source: https://doi.org/10.1080/19434471003597415

Emotional contagion on Facebook

Kramer et al. (2014): study of 689,003 Facebook users

Experimentally manipulated content shown on news feeds to test **emotional contagion hypothesis**

- ➜ Treatment 1: Positive content more visible on news feed
- ➜ Treatment 2: Negative content more visible on news feed
- ➜ Control: No news feed intervention

Use ADM to measure sentiment of users' Facebook posts post-treatment, showing:

- ➜ positive content boosts positive posting, reduces negative
- ➜ negative content boosts negative posting, reduces positive

Concerns about ethics of the study, see this editorial statement

Source: https://www.pnas.org/doi/10.1073/pnas.1320040111

Validating a dictionary

Automated dictionary methods should always be validated!

Qualitative evaluations:

- ➜ When creating or choosing a dictionary: careful identification of concepts, associated keys/categories, and textual features associated with each key/category
- ➜ Read some documents to get a gut check about whether the dictionary is capturing meanings correctly
- ➜ Check sensitivity of results to exclusion of specific words

Validating a dictionary

Automated dictionary methods should always be validated!

Quantitative evaluations:

- ➜ Create a human-coded **validation set** and compare your dictionary method, treating human-coding as the truth
  - ➜ **Accuracy**: what % of documents is correctly coded by your ADM?
  - ➜ **Precision** of category: what % of documents coded in a category by your ADM were coded correctly?
  - ➜ **Recall**: what % of documents coded in a category by human-coders were coded correctly by your ADM?

(We'll cover this in more detail next week)

Weakness: context specificity

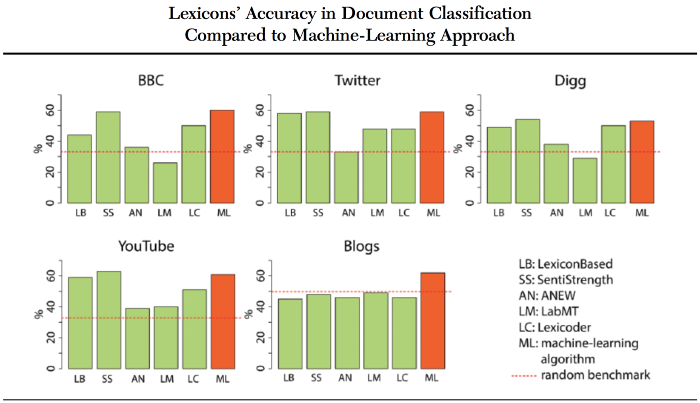
Loughran and McDonald (2011): used the Harvard-IV-4 TagNeg (H4N) file to classify sentiment for a corpus of 50,115 firm-year 10-K filings from 1994–2008

- Almost three-fourths of “negative” words in dictionary were not typically negative in financial context, e.g. `tax`, `cost`, `crude`, `cancer`
- Problem is **polysemes**—words that have multiple meanings
- Another problem: dictionary lacked important negative financial words, such as `felony`, `litigation`, `restated`, `misstatement`, and `unanticipated`

Source: https://doi.org/10.1111/j.1540-6261.2010.01625.x

Weakness: context specificity

González-Bailón and Paltoglou (2015) shows how different dictionaries perform differently across different contexts:



Source: https://doi.org/10.1177/0002716215569192

Weakness: worse than ML classification

Barberá et al. (2021) codes a set of >4,000 articles, showing ML generally does better than ADMs coding sentiment

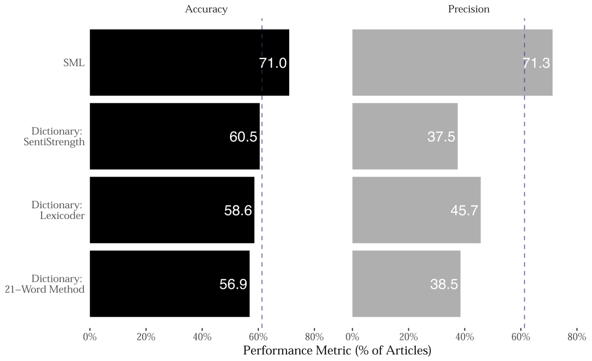


Figure 3. Performance of SML and Dictionary Classifiers—Accuracy and Precision.
Note: Accuracy (percent correctly classified) and precision (percent of positive predictions that are correct) for the ground truth dataset coded by ten CrowdFlower coders. The dashed vertical lines indicate the baseline level of accuracy or precision (on any category) if the modal category is always predicted. The corpus used in the analysis is based on the keyword search of *The New York Times* 1980–2011 (see the text for details).

Source: http://doi.org/10.1017/pan.2020.8

Weakness: sensitive to frequent words

Pury (2011) showing problem with ADM in Back et al. (2010):

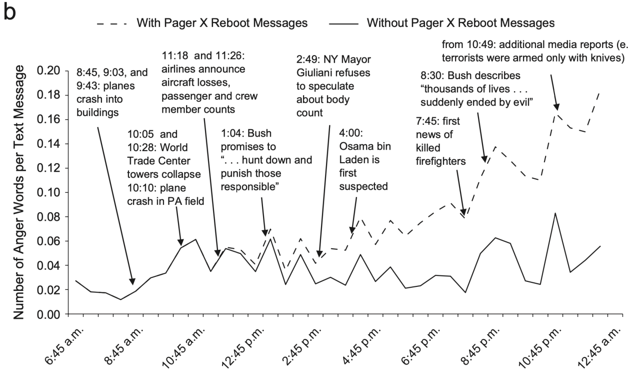


Fig. 1. Proportion of automatically generated text messages and timeline of anger as expressed in text messages on September 11, 2001. The graph in (a) shows the proportion of the analyzed pager messages that were nearly identical technical "reboot" messages sent to a single pager, Pager X, as a function of time block. The graph in (b) shows the mean number of Linguistic Inquiry and Word Count anger words per alphanumeric pager message as a function of time block; means were calculated both with and without the Pager X messages included in the population of messages. In both graphs, the 30-min time blocks are indicated by their starting time. The timeline in (b) is based on Figure 1 in Back, Küfner, and Egloff (2010).

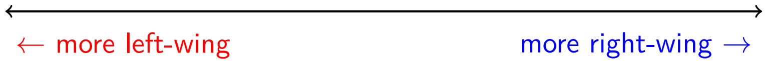
Sources: original paper, https://doi.org/10.1177/0956797610382124; critique, https://doi.org/10.1177/0956797611408735; reanalysis, https://doi.org/10.1177/0956797611409592

Scaling

What is scaling?

Scaling is a set of quantitative tools for measuring **latent traits**, which are typically measured on continuous "scales"

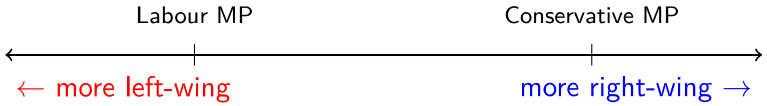
- ➜ Classic example: political ideology on "left-right" scale



What is scaling?

Scaling is a set of quantitative tools for measuring **latent traits**, which are typically measured on continuous "scales"

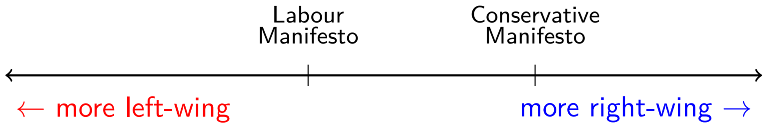
- Classic example: political ideology on "left-right" scale
  - In political science: lots of research placing individual politicians on ideological scales



What is scaling?

Scaling is a set of quantitative tools for measuring **latent traits**, which are typically measured on continuous "scales"

- ➜ Classic example: political ideology on "left-right" scale
  - ➜ Also—lots of research placing *documents* on ideological scales as well (Recall: texts are observable implications)



What is scaling?

But that’s not all...

- Policy positions on economic vs social dimension
- Inter- and intra-party differences
- Soft news vs hard news
- Petitioner vs respondent in legal briefs
- ...and any other continuous scale

Two major approaches: supervised versus unsupervised

- We’ll cover supervised: use labelled “reference” texts to learn and apply to unlabelled texts

Scaling versus ADMs

ADMs: predefined list of words that represent useful "concepts", such as positive versus negative sentiment

- ➜ Then, just measure use of these pre-defined words in documents and apply a weighting scheme

Supervised scaling: same as a dictionary method *except* you do not need to start with a predefined list of words

- ➜ Use categories of interest (e.g., partisanship) to create the word weights in $\boldsymbol{\mu}_k$ directly (bypass the dictionary)

Similar in spirit to classification (next week)

Supervised scaling with Wordscores

Wordscores method by Laver, Benoit and Garry (2003):

- ➜ Start with set of texts with *known* positions on one (or more) "left-right" scale
- ➜ Use these texts to measure the so-called **word scores** for each word used in these texts
- ➜ Use these word scores to create **document scores** for different set of texts with unknown positions

Very similar procedure to classification!

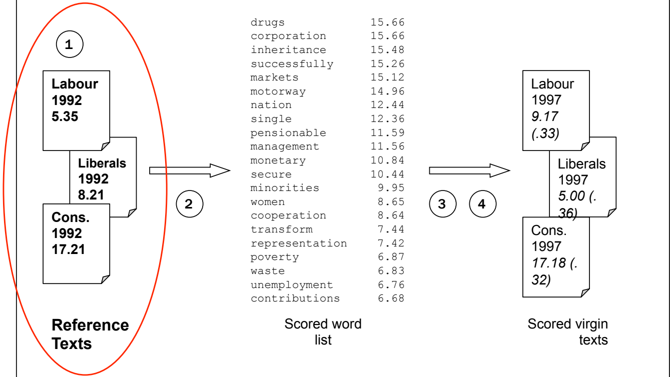
Running example: corpus of 1992 and 1997 UK manifestos

- ➜ Sourced from the `{quanteda}` package in R

The Wordscores procedure

The Wordscore Procedure

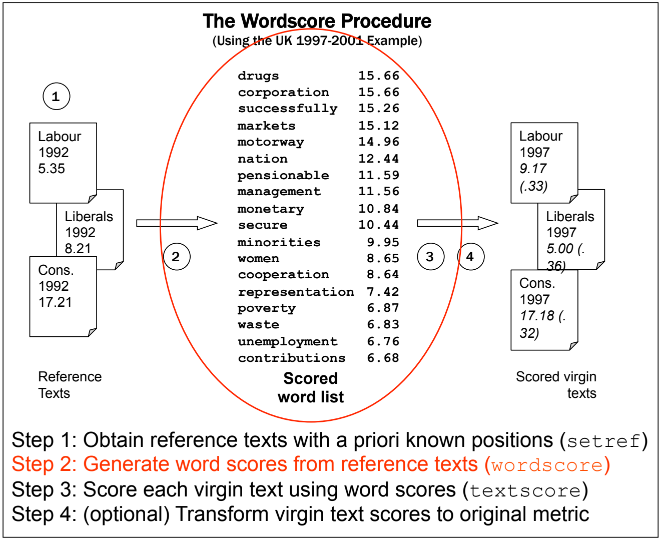
(Using the UK 1997-2001 Example)



drugs	15.66
corporation	15.66
inheritance	15.48
successfully	15.26
markets	15.12
motorway	14.96
nation	12.44
single	12.36
pensionable	11.59
management	11.56
monetary	10.84
secure	10.44
minorities	9.95
women	8.65
cooperation	8.64
transform	7.44
representation	7.42
poverty	6.87
waste	6.83
unemployment	6.76
contributions	6.68

Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

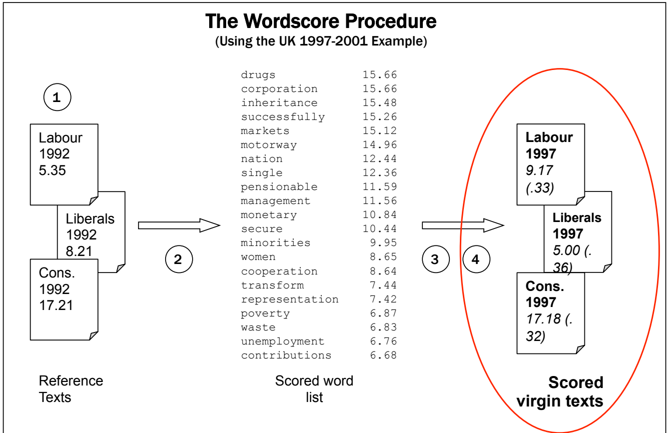
The Wordscores procedure



drugs	15.66
corporation	15.66
successfully	15.26
markets	15.12
motorway	14.96
nation	12.44
pensionable	11.59
management	11.56
monetary	10.84
secure	10.44
minorities	9.95
women	8.65
cooperation	8.64
representation	7.42
poverty	6.87
waste	6.83
unemployment	6.76
contributions	6.68

Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

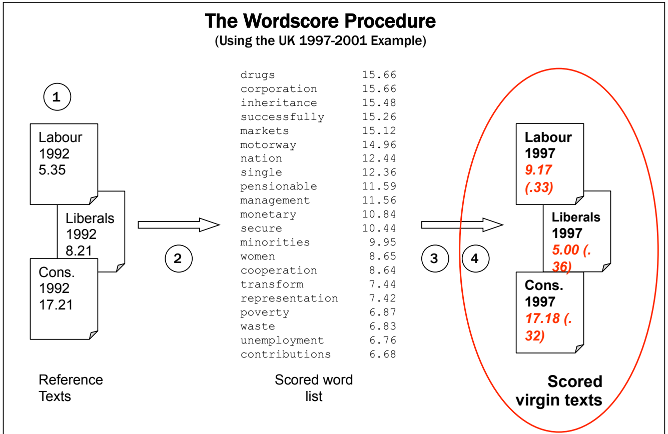
The Wordscores procedure



drugs	15.66
corporation	15.66
inheritance	15.48
successfully	15.26
markets	15.12
motorway	14.96
nation	12.44
single	12.36
pensionable	11.59
management	11.56
monetary	10.84
secure	10.44
minorities	9.95
women	8.65
cooperation	8.64
transform	7.44
representation	7.42
poverty	6.87
waste	6.83
unemployment	6.76
contributions	6.68

Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

The Wordscores procedure



drugs	15.66
corporation	15.66
inheritance	15.48
successfully	15.26
markets	15.12
motorway	14.96
nation	12.44
single	12.36
pensionable	11.59
management	11.56
monetary	10.84
secure	10.44
minorities	9.95
women	8.65
cooperation	8.64
transform	7.44
representation	7.42
poverty	6.87
waste	6.83
unemployment	6.76
contributions	6.68

Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

Calculating Wordscores: prerequisites

- We'll use the 1992 documents to create word scores
  - The **labelled set** (or "reference texts" in LBG 2003)
- Then we'll apply estimated word scores to 1997 documents to measure their ideology
  - The **unlabelled set** (or "virgin texts" in LBG 2003)
- We'll assume these document scores for the labelled set:
  - Conservative 1992: $\pi_{C92} = 17.21$
  - Labour 1992: $\pi_{L92} = 5.35$
  - Liberal Democrat 1992: $\pi_{LD92} = 8.21$
- Where do these come from? Expert coding...

Calculating Wordscores: prerequisites

```python
import pandas as pd
import numpy as np
df = pd.read_csv("UK_Manifestos.csv", dtype = "object")
print(df)
```

```
    docname                                               text
0  Con_1992  Conservative Party 1992 \nThe Best Future for ...
1  Lab_1992  \nTHE LABOUR PARTY MANIFESTO\n\nTIME TO GET BR...
2   LD_1992  1992\nCHANGING BRITAIN FOR GOOD\n\nAFTER THE H...
3  Con_1997  THE CONSERVATIVE MANIFESTO 1997\n\nFOREWORD\nT...
4  Lab_1997  BRITAIN WILL BE BETTER WITH NEW LABOUR\n\n'OUR...
5   LD_1997  make the\ndifference\n\nThe Liberal Democrat M...
```

```python
pos = np.array([[17.21, 5.35, 8.21]])  # positions for labelled
pos
```

```
array([[17.21,  5.35,  8.21]])
```

```python
ltexts = [0,1,2]           # indices for labelled
utexts = [3,4,5]           # indices for unlabelled
```

Calculating Wordscores: pre-processing

Convert the texts into a document-feature matrix

- ➜ You do not need to do stemming or remove stop words, but you can if you want (see Lowe and Benoit 2013)
- ➜ Just be consistent and do what makes sense for your task

We will use the usual notation we've developed so far:

- ➜ $\mathbf{W}$ is an $N \times J$ document feature matrix,
- ➜ $\mathbf{W}_i$ is a specific document (row) of the DFM
- ➜ $M_i = \sum_{j=1}^{J} W_{ij}$ is the document length for document i

Calculating Wordscores: pre-processing

```python
import re
from collections import Counter
from sklearn.feature_extraction import DictVectorizer

def preprocess_tokens(toks):
    toks = [x.lower() for x in toks]
    toks = [re.sub(r"[^\w\-\']", "", x) for x in toks]
    toks = [re.sub(r"(^[\-\']|[\-\']$)", "", x) for x in toks]
    toks = [x for x in toks if not re.search(r"[^A-Za-z\-\']", x)]
    toks = [x for x in toks if not x in [""]]
    return toks

df["pp"] = df["text"].str.split(r"\s+")
df["pp"] = df["pp"].apply(preprocess_tokens)

dv = DictVectorizer()
dfm = dv.fit_transform(df["pp"].map(Counter).to_list())
vocabulary = dv.get_feature_names_out()
```

Calculating Wordscores: pre-processing

We're going to trim any feature not occurring at least once in the labelled documents

```
ttfl = dfm[ltexts].sum(axis = 0).A1
dfm = dfm[:,(ttfl >= 1)]
vocabulary = vocabulary[(ttfl >= 1)]
dfm.shape
```

```
(6, 5471)
```

Calculating Wordscores: pre-processing

We can preview the DFM—just five interesting features

```
          drugs  women  markets  poverty  contributions
docname
Con_1992      9     24       12        1              0
Lab_1992      0     23        0        5              3
LD_1992       1     18        2        4              4
Con_1997     11     13        7        3              5
Lab_1997      4      9        4       10              2
LD_1997       2     11        1       10              5
```

Calculating Wordscores: normalise DFM

Begin the estimation process by normalising the DFM

- ➜ Create a **relative document-feature matrix** $\mathbf{F}$, by dividing each W_{ij} by its **word total marginals**
- ➜ In other words, divide each row by the sum of that row

So, each element of $\mathbf{F}$ is calculated by

$$F_{ij} = \frac{W_{ij}}{M_i} = \frac{W_{ij}}{\sum_j W_{ij}}$$

Calculating Wordscores: normalise DFM

```
F = dfm.multiply(1 / dfm.sum(axis=1))
F = F.tocsr()
```

We can preview the $\mathbf{F}$ matrix:

```
            drugs    women  markets  poverty  contributions
docname
Con_1992  0.00031  0.00081  0.00041  0.00003        0.00000
Lab_1992  0.00000  0.00203  0.00000  0.00044        0.00026
LD_1992   0.00006  0.00105  0.00012  0.00023        0.00023
Con_1997  0.00056  0.00066  0.00035  0.00015        0.00025
Lab_1997  0.00024  0.00055  0.00024  0.00061        0.00012
LD_1997   0.00015  0.00082  0.00007  0.00074        0.00037
```

Calculating Wordscores: document probs. in labeled set

Using $\mathbf{F}$, next we compute the **relative document probabilities** for each of the *labelled* texts

- ➜ Let L be the set of labelled documents, of which there are N^L

Create an $N^L \times J$ matrix $\mathbf{P}$, where each element P_{ij} is:

$$P_{ij} = \frac{F_{ij}}{F_j} = \frac{F_{ij}}{\sum_i F_{ij}} \text{ for all } i \in L$$

- ➜ For each document i and word j, P_{ij} is the probability that we are reading a certain reference document i if it contains j
- ➜ Note that $\mathbf{P}$ only contains data for the labelled documents

Calculating Wordscores: document probs. in labelled set

```
P = F[ltexts].multiply(1 / F[ltexts].sum(axis=0))
P = P.tocsr()
```

We can preview the $\mathbf{P}$ matrix:

```
             drugs    women  markets  poverty  contributions
docname
Con_1992  0.84004  0.20932  0.77783  0.04794        0.00000
Lab_1992  0.00000  0.52164  0.00000  0.62339        0.53228
LD_1992   0.15996  0.26904  0.22217  0.32866        0.46772
```

Calculating Wordscores: calculate word scores

Using $\mathbf{P}$, we can compute a J-length vector S that gives the "word score" for each word

Specifically, each element in the vector is calculated as follows:

$$s_j = \sum_i (\pi_i \times P_{ij})$$

➜ s_j is the average of each document i's scores π_i, weighted by each word's P_{ij}

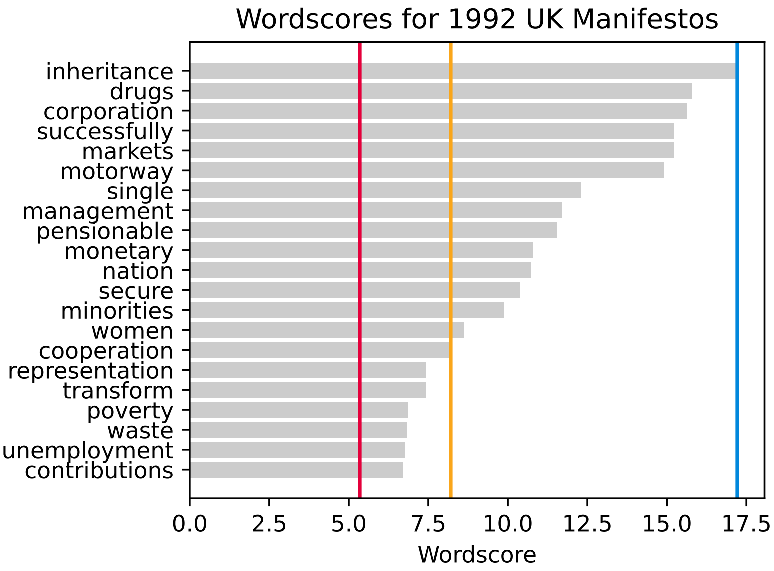
Calculating Wordscores: calculate word scores

```
S = P.multiply(pos.T)
S = S.tocsr()
S = S.sum(axis=0)
```

```
wordscores = pd.Series(np.asarray(S)[0], index=vocabulary)
wordscores = wordscores.sort_values()
wordscores
```

```
certainly                    5.35
continuation                 5.35
undertakes                   5.35
bcworldservicepraisedby      5.35
contentious                  5.35
                             ...
extends                     17.21
extensions                  17.21
extensive                   17.21
recasting                   17.21
zone                        17.21
Length: 5471, dtype: float64
```

Calculating Wordscores: calculate word scores



Calculating Wordscores for all texts

Now, we obtain a *single* estimated document score for each text:

$$\hat{\pi}_i = \sum_j (F_{ij} \times s_j)$$

In Python:

```python
pi = F.multiply(S).sum(axis = 1)
pi
```

```
matrix([[11.28340138],
        [ 9.5107655 ],
        [ 9.97583312],
        [10.74123501],
        [10.39706775],
        [10.22663097]])
```

Calculating Wordscores: rescale scores

Unfortunately, the estimated scores aren't on same scale as the original positions

- ➜ Notice: estimates are more "bunched" together
- ➜ This is because there are a lot of non-discriminating words pulling the estimates toward the middle

We can rescale them in a couple ways:

- ➜ Original LBG paper proposes one way
- ➜ Martin and Vanberg (2007) propose another

Won't get into details, but both involve affine transformations:

$$\hat{\pi}_i^{\mathrm{r}} = a + b\hat{\pi}_i$$

Calculating Wordscores: rescale scores

Create a data frame to collect raw estimates plus rescaled

```
ds = pd.DataFrame(pi, columns = ["raw"])
ds = pd.concat([df["docname"], pd.Series(pos[0,:]), ds], axis = 1)
ds.columns = ["docname", "apriori", "raw"]
print(ds)
```

```
    docname  apriori        raw
0  Con_1992    17.21  11.283401
1  Lab_1992     5.35   9.510766
2   LD_1992     8.21   9.975833
3  Con_1997      NaN  10.741235
4  Lab_1997      NaN  10.397068
5   LD_1997      NaN  10.226631
```

Calculating Wordscores: rescale scores

LBG rescaling in Python:

```
ref = ds["apriori"].notna()
ds["lbg"] = ds["raw"] - ds.loc[ref, "raw"].mean()
ds["lbg"] = ds["lbg"] * ds.loc[ref, "apriori"].std(ddof=1)
ds["lbg"] = ds["lbg"] / ds.loc[ref, "raw"].std(ddof=1)
ds["lbg"] = ds["lbg"] + ds.loc[ref, "apriori"].mean()
print(ds)
```

```
    docname  apriori        raw        lbg
0  Con_1992    17.21  11.283401  17.170849
1  Lab_1992     5.35   9.510766   5.233659
2   LD_1992     8.21   9.975833   8.365492
3  Con_1997      NaN  10.741235  13.519821
4  Lab_1997      NaN  10.397068  11.202148
5   LD_1997      NaN  10.226631  10.054402
```

Calculating Wordscores: rescale scores

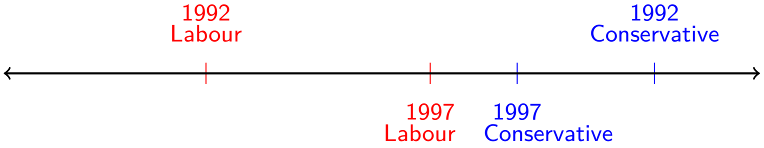
Martin-Vanberg rescaling in Python:

```
i1 = ds.loc[ref, "apriori"].idxmin()
i2 = ds.loc[ref, "apriori"].idxmax()

ds["mv"] = ds["raw"] - ds.loc[i1, "raw"]
ds["mv"] = ds["mv"] * (ds.loc[i2, "apriori"] - ds.loc[i1, "apriori"])
ds["mv"] = ds["mv"] / (ds.loc[i2, "raw"] - ds.loc[i1, "raw"])
ds["mv"] = ds["mv"] + ds.loc[i1, "apriori"]
print(ds)
```

```
    docname  apriori        raw        lbg         mv
0  Con_1992    17.21  11.283401  17.170849  17.210000
1  Lab_1992     5.35   9.510766   5.233659   5.350000
2   LD_1992     8.21   9.975833   8.365492   8.461582
3  Con_1997      NaN  10.741235  13.519821  13.582581
4  Lab_1997      NaN  10.397068  11.202148  11.279895
5   LD_1997      NaN  10.226631  10.054402  10.139570
```

Calculating Wordscores: rescale scores



Calculating Wordscores: other considerations

You can do a couple other things:

1. You will likely want to provide confidence intervals for your estimates on the unlabeled set
   - ➜ Can use an analytical approach (as in LBG 2003)
   - ➜ Or you can bootstrap (as in Lowe and Benoit 2013)
2. You may wish to regularise the labelled portion of the DFM to avoid zeroes that distort word score estimation

Getting labelled texts

What kind of considerations go into choosing labelled texts?

1. Labelled set should contain texts that clearly represent an *a priori* known dimension
2. Labelled texts should be as discriminating as possible, e.g. “extreme” texts
3. Labelled texts must contain lots of words
4. Labelled & unlabelled texts must come from same lexical universe

What about the labels (i.e., positions on scale)? Some latitude here, but most important consideration is use case:

- ➜ Choose labels on pre-existing scale for interpretability?
- ➜ Can always rescale, e.g. from –1 to 1