Week 4: Dictionaries

POP77032 Quantitative Text Analysis for Social Scientists

Tom Paskhalis

Overview

- Bag-of-words
- Document-Term Matrix
- Dictionaries
- Text classification

Bag-of-words

Quantifying Texts

When I presented the supplementary budget to this House last April, I said we could work our way through this period of severe economic distress. Today, I can report that notwithstanding the difficulties of the past eight months, we are now on the road to economic recovery.

In this next phase of the Government's plan we must stabilise the deficit in a fair way, safeguard those worst hit by the recession, and stimulate crucial sectors of our economy to sustain and create jobs. The worst is over.

This Government has the moral authority and the well-grounded optimism rather than the cynicism of the Opposition. It has the imagination to create the new jobs in energy, agriculture, transport and construction that this green budget will

	words										
docs	made	because	had	into	get	some	through	next	where	many	irish
t06_kenny_fg	12	11	5	4	8	4	3	4	5	7	10
t05_cowen_ff	9	4	8	5	5	5	14	13	4	9	8
t14_ocaolain_sf	3	3	3	4	7	3	7	2	3	5	6
t01_lenihan_ff	12	1	5	4	2	11	9	16	14	6	9
t11_gormley_green	0	0	0	3	0	2	0	3	1	1	2
t04_morgan_sf	11	8	7	15	8	19	6	5	3	6	6
t12_ryan_green	2	2	3	7	0	3	0	1	6	0	0
t10_quinn_lab	1	4	4	2	8	4	1	0	1	2	0
t07_odonnell_fg	5	4	2	1	5	0	1	1	0	3	0
t09_higgins_lab	2	2	5	4	0	1	0	0	2	0	0
t03_burton_lab	4	8	12	10	5	5	4	5	8	15	8
t13_cuffe_green	1	2	0	0	11	0	16	3	0	3	1
t08_gilmore_lab	4	8	7	4	3	6	4	5	1	2	11
t02_bruton_fg	1	10	6	4	4	3	0	6	16	5	3

Descriptive statistics on words

Scaling documents

Classifying documents

Extraction of topics

Vocabulary analysis

Sentiment analysis

Tokenisation

- Say, we *tokenised* a single text:

```
text <- "The quick brown fox jumps over the lazy dog."
tokens <- quanteda::tokens(text)
tokens
```

```
Tokens consisting of 1 document.
text1 :
 [1] "The"   "quick" "brown" "fox"   "jumps" "over"  "the"   "lazy"  "dog"
[10] "."
```

- But what if we have multiple texts?

Document-Term Matrix (DTM)

- Imagine in addition to the text above, we have another one:

```
text2 <- "The quick brown fox jumps over the lazy cat."
```

- If we were to create a **document-term matrix (DTM)**, it would look like this:

Document	brown	cat	dog	fox	jumps	lazy	over	quick	the
1	1	0	1	1	1	1	1	1	2
2	1	1	0	1	1	1	1	1	2

- Where each row corresponds to a document and each column to a token.
- In math terms, a matrix $\mathbf{W}$ of N documents (rows) and J terms (columns),
  - where each W_{ij} shows the number of times the jth term appears in the ith document.
  - also referred to as the **term frequency** of term j in document i.

DTM

- In real life the documents are unlikely to have as much overlap.
- A more realistic example of 2 texts would be something like this:

```
speech1 <- "I thank the Deputy."
speech2 <- "Deputy, please resume your seat."
```

- If we we were to create a document-term matrix, it would look like this:

Document	deputy	i	please	resume	seat	thank	the	your
1	1	1	0	0	0	1	1	0
2	1	0	1	1	1	0	0	1

- This is quite a few zeros!

Sparse Matrix

- In practice, DTMs are often stored as **sparse matrices**.
- Such matrix only stores non-zero values and their positions.

```r
library("Matrix")
sm <- Matrix::sparseMatrix(
  i = c(1, 1, 1, 1, 2, 2, 2, 2, 2),
  j = c(1, 2, 6, 7, 1, 3, 4, 5, 8),
  x = c(1, 1, 1, 1, 1, 1, 1, 1, 1),
  dims = c(2, 8),
  dimnames = list(
    c("Doc1", "Doc2"),
    c("deputy", "i", "please", "resume", "seat", "thank", "the", "your"))
)
sm
```

```
2 x 8 sparse Matrix of class "dgCMatrix"
     deputy i please resume seat thank the your
Doc1      1 1      .      .    .     1   1    .
Doc2      1 .      1      1    1     .   .    1
```

- Of course, we will not be creating these matrices manually.

DTM in R

- In R we can use the `quanteda` package to create DTMs.

```
library("quanteda")
```

- We start by tokenising our small corpus:

```
speeches <- c(speech1, speech2)
speeches_toks <- quanteda::tokens(tolower(speeches), remove_punct = TRUE)
speeches_toks
```

```
Tokens consisting of 2 documents.
text1 :
[1] "i"      "thank"  "the"    "deputy"

text2 :
[1] "deputy" "please" "resume" "your"   "seat"
```

- Then we create a DTM (or, as it is called here, a **document-feature matrix (DFM)**):

```
speeches_dfm <- quanteda::dfm(speeches_toks)
speeches_dfm
```

```
Document-feature matrix of: 2 documents, 8 features (43.75% sparse) and 0 docvars.
       features
docs    i thank the deputy please resume your seat
  text1 1     1   1      1      0      0    0    0
  text2 0     0   0      1      1      1    1    1
```

DTM in Python

- In Python we can use the ML library `sklearn` to work with DTMs.

```
1 import sklearn
```

```
1 speeches = ["I thank the Deputy.", "Deputy, please resume your seat."]
2 vectorizer = sklearn.feature_extraction.text.CountVectorizer(token_pattern = r"\b\w+\b")
3 dtm = vectorizer.fit_transform(speeches)
4 dtm
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
    with 9 stored elements and shape (2, 8)>
```

- To see the tokenised features:

```
1 vectorizer.get_feature_names_out()
```

```
array(['deputy', 'i', 'please', 'resume', 'seat', 'thank', 'the', 'your'],
      dtype=object)
```

- To see the DTM as an array:

```
1 dtm.toarray()
```

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

Bag of Words

- This representation of text is also known as **bag-of-words**.
- The key aspects of this approach are:
  - Single words are considered as relevant *features* of each document.
  - Documents are quantified by counting occurrences of words.
  - Word order is ignored.
  - Grammar and syntax are discarded.

Why Bag of Words?

- Simple.
- Efficient.
- Works well in many cases.
- Can be extended (co-occurrences aka “n-grams”)
- Has been extensively used and validated in social sciences.
- Of course, there are cases when word order matters (e.g. text reuse).

Trimming DTMs

- As we have seen, by default, most DTMs tend to be very sparse.
- This can be a problem as this results in computational inefficiencies.
- It might also be the case that not tokens are equally informative.
- Some examples of trimming features that we have already encountered:
  - removing punctuation
  - removing stopwords
  - creating **equivalence classes** through stemming/lemmatisation
- A more structured way would be to trim by frequency

Trimming by Frequency

- **Document frequency** of term j counts how many documents contain the feature j:

$$\mathrm{DF}_j = \sum_{i=1}^{N} \mathbb{1}(W_{ij} > 0)$$

- **Total term frequency** of term j counts how many times the feature j appears in the corpus:

$$\mathrm{TTF}_j = \sum_{i=1}^{N} W_{ij}$$

- We can, thus, decide to trim certain tokens that are too uncommon.
  - e.g. don't appear in at least 3 documents at least 5 times.

Rare vs Frequenct Terms

- Deciding which features to keep is a guesstimate about their informativeness.
- So far we removed the words that appear too rarely to carry meaningful information.
- But this leaves with a lot of very common words that are also not very informative.
- Some of those could be removed with *stopwords* lists.
- But stopwords lists are context independent and might not fit our specific task.

Weighting DTMs

- One way to look at trimming is a form of **weighting**.
- What trimming does is to assign a weight of 0 to certain features.
- But this this is a rather crude way to weight features.
- Instead what if we could devise a weighting scheme that would:
  - Downweight rare and uninformative features;
  - Downweight common and uninformative features;
  - Upweight more informative features.

TF-IDF Weighting

- Most common weighting schemes is **term frequency-inverse document frequency (TF-IDF)**.
- Formally, the simplest version of TF-IDF weighting would then be:

$$\text{TF-IDF}_{ij} = \text{TF}_{ij} \times \text{IDF}_j = \text{TF}_{ij} \times \frac{N}{\text{DF}_j}$$

- Effectively, what TF-IDF does is to:
  - upweight terms that are frequent in a document but rare across the corpus.
  - downweight terms that are either rare in a document or common across the corpus.

Varieties of TF-IDF Weighting

- We considered the most basic version of TF-IDF weighting.
- One of the most common versions takes the logarithm of the IDF component:

$$\text{TF-IDF}_{ij} = \text{TF}_{ij} \times \log \frac{N}{\text{DF}_j}$$

- This version is more robust to very rare terms that would otherwise get very high weights.
- E.g. the default version in `sklearn` prevents division-by-zero (if unused tokens are not removed) and softens extremes by rescaling the IDF term:

$$\text{TF-IDF}_{ij} = \text{TF}_{ij} \times (\log \frac{1+N}{1+\text{DF}_j} + 1)$$

Extra

Scikit-learn documentation on TF-IDF weighting

TF-IDF Weighting in R

- In R we can use the `dfm_tfidf()` function to apply TF-IDF weighting to our DFM.
- Note that we keep all the arguments to the default, but it's something you might want to adjust depending on your task:.

```
1 tfidf_dfm <- quanteda::dfm_tfidf(speeches_dfm)
2 tfidf_dfm
```

```
Document-feature matrix of: 2 documents, 8 features (43.75% sparse) and 0 docvars.
       features
docs          i   thank     the deputy  please  resume    your    seat
  text1 0.30103 0.30103 0.30103      0 0       0       0       0
  text2 0       0       0            0 0.30103 0.30103 0.30103 0.30103
```

- Compare it to the original DFM:

```
1 speeches_dfm
```

```
Document-feature matrix of: 2 documents, 8 features (43.75% sparse) and 0 docvars.
       features
docs    i thank the deputy please resume your seat
  text1 1     1   1      1      0      0    0    0
  text2 0     0   0      1      1      1    1    1
```

- E.g. $\text{TD-IDF}_i = \text{TF}_{ij} \times \log_{10} \frac{N}{\text{DF}_j} = 1 \times log_{10} \frac{2}{1} = 0.30103$ for the feature i.

TF-IDF Weighting in Python

- In Python we can use the `TfidfTransformer` class to to the same effect.
- Note that the default scheme can vary across implementations, do check the documentation for the exact formula.

```
1 tfidf_transformer = sklearn.feature_extraction.text.TfidfTransformer()
2 tfidf_dtm = tfidf_transformer.fit_transform(dtm)
```

```
1 vectorizer.get_feature_names_out()
```

```
array(['deputy', 'i', 'please', 'resume', 'seat', 'thank', 'the', 'your'],
      dtype=object)
```

```
1 tfidf_dtm.toarray()
```

```
array([[0.37997836, 0.53404633, 0.        , 0.        , 0.        ,
        0.53404633, 0.53404633, 0.        ],
       [0.33517574, 0.        , 0.47107781, 0.47107781, 0.47107781,
        0.        , 0.        , 0.47107781]])
```

Dictionaries

Word Meanings

- Words have meanings 🧐
- This allows us to take word usage as a proxy for the overall ‘meaning’ of a text.
- Certain kinds of words indicate certain kind of ‘meanings’.
- Kinds of ‘meanings’:
  - Sentiment (e.g. positive, negative, etc.)
  - Emotions (e.g. anger, sad, happiness, etc.)
  - Topics (e.g. politics, sports, etc.)
  - Ideology (e.g. liberal, conservative, etc.)
  - Hate speech (e.g. sexism, homophobia, xenophobia, etc.)

Dictionaries

- **Automated dictionary methods (ADM)** exploit word usage to learn the ‘meanings’ of texts.
- Two steps:
  1. **Dictionary creation**: Define a list of words that represent a certain ‘meaning’.
  2. **Dictionary application**: Count the number of words in a text that are in the dictionary.
- Dictionaries should be task-appropriate and validated.

Dictionary Structure

- We have seen dictionaries in the context of Python:

```
ideo_dict = {
  "liberal": ["benefits", "worker", "trade union"],
  "conservative": ["restriction", "immigration", "reduction"]
  }
```

- Essentially, a dictionary is a set of **key-value pairs**.
- In the context of text analysis:
  - **Keys** - labels for equivalence classes for the concept of interest.
  - **Values** - terms or patterns that are declared equivalent occurrences of the key class

Dictionary vs Thesaurus

- A *dictionary* in a QTA sense is somewhat of a misnomer.
- Substantively, a dictionary is closer to a *thesaurus*.
- I.e. a list of canonical terms or concepts ('keys') associated with a list of synonyms.
- But unlike thesauruses, ADM dictionaries:
  - tend to be 'exclusive' (each value is associated with one key only)
  - do not always identify synonyms

Qualitative & Quantitative Text Analysis

- ADM dictionaries sit somewhere between more qualitative and fully automated approaches to text analysis.
- It is ‘qualitative’ in a sense that it requires identification of concepts and textual features associated with each of them.
- Dictionary construction involves a lot of contextual interpretation and qualitative judgment
- At the same time the application part is fully automated and perfectly reliable/replicable.

Some Famous Dictionaries

- **General Inquirer** (Stone et al. 1966): an early all-purpose dictionary (e.g. sentiment analysis) in general texts.
- **Regressive Imagery Dictionary**: designed to measure primordial vs. conceptual thinking.
- **Linguistic Inquiry and Word Count (LIWC)** (Pennebaker et al. 2001): large (paid) dictionary for many psychological and related concepts.

Example: LexiCoder

- The **LexiCoder Sentiment Dictionary** (Young and Soroka 2012): a dictionary for sentiment analysis in political texts, validated with human-coded news content.

```
data("data_dictionary_LSD2015", package = "quanteda.dictionaries")
```

```
str(data_dictionary_LSD2015)
```

```
Formal class 'dictionary2' [package "quanteda"] with 2 slots
  ..@ .Data:List of 4
  .. ..$ :List of 1
  .. .. ..$ : chr [1:2858] "a lie" "abandon*" "abas*" "abattoir*" ...
  .. ..$ :List of 1
  .. .. ..$ : chr [1:1709] "ability*" "abound*" "absolv*" "absorbent*" ...
  .. ..$ :List of 1
  .. .. ..$ : chr [1:1721] "best not" "better not" "no damag*" "no no" ...
  .. ..$ :List of 1
  .. .. ..$ : chr [1:2860] "not a lie" "not abandon*" "not abas*" "not abattoir*" ...
  ..@ meta :List of 3
  .. ..$ system:List of 5
  .. .. ..$ package-version:Classes 'package_version', 'numeric_version'  hidden list of 1
  .. .. .. ..$ : int [1:3] 1 9 9009
  .. .. ..$ r-version      :Classes 'R_system_version', 'package_version', 'numeric_version'  hidden list of 1
  .. .. .. ..$ : int [1:3] 3 6 2
  .. .. ..$ system         : Named chr [1:3] "Darwin" "x86_64" "kbenoit"
  .. .. .. ..- attr(*, "names")= chr [1:3] "sysname" "machine" "user"
  .. .. ..$ directory      : chr "/Users/kbenoit/Dropbox (Personal)/GitHub/quanteda/quanteda"
  .. .. ..$ created        : Date[1:1], format: "2020-02-17"
```

Example: Laver and Garry (2000)

- A *hierarchical* set of categories to distinguish policy domains and policy positions.
- Derived from one of the longest content analysis exercises in political science - Manifesto Project (previously known as CMP).
- Five domains at the top level of hierarchy:
  - economy
  - political system
  - social system
  - external relations
  - "general" domain
- The dictionary was developed on a set of specific UK manifestos.

Extra

(Laver & Garry, 2000)

Example: Laver and Garry (2000)

- An accompanying `quanteda.dictionaries` package contains a lot of mentioned dictionaries, including the Laver and Garry (2000) dictionary.
- Alternatively, you can download the 'raw' dictionary as text from https://provalisresearch.com/Download/LaverGarry.zip

```
# The package is not available on CRAN,
# so need to install it from GitHub
remotes::install_github("kbenoit/quanteda.dictionaries")
```

```
data("data_dictionary_LaverGarry", package = "quanteda.dictionaries")
```

```
str(data_dictionary_LaverGarry)
```

```
Formal class 'dictionary2' [package "quanteda"] with 2 slots
  ..@ concatenator: chr " "
  ..@ names       : chr [1:9] "CULTURE" "ECONOMY" "ENVIRONMENT" "GROUPS" ...
  ..@ .Data       :List of 9
  .. ..$ :List of 4
  .. .. ..$ CULTURE-HIGH   :List of 1
  .. .. .. ..$ : chr [1:8] "art" "artistic" "dance" "galler*" ...
  .. .. ..$ CULTURE-POPULAR:List of 1
  .. .. .. ..$ : chr "media"
  .. .. ..$ SPORT          :List of 1
  .. .. .. ..$ : chr "angler*"
  .. .. ..$                : chr [1:3] "people" "war_in_iraq" "civil_war"
  .. ..$ :List of 3
  .. .. ..$ +STATE+:List of 1
  .. .. .. ..$ : chr [1:50] "accommodation" "age" "ambulance" "assist" ...
```

```
.. .. ..$ =STATE=:List of 1
.. .. .. ..$ : chr [1:71] "accountant" "accounting" "accounts" "advert*" ...
.. .. ..$ -STATE-:List of 1
.. .. .. ..$ : chr [1:62] "assets" "autonomy" "barrier*" "bid" ...
.. ..$ :List of 2
.. .. ..$ CON ENVIRONMENT:List of 1
```

Example: Dictionary Application

- Imagine we want to know which of the parties discusses immigration the most in their electoral manifesto.
- We can start by creating a very simple dictionary to answer this question:

```
imm_dict <- quanteda::dictionary(list(
  immigration = c("asylum*", "border*", "immigra*", "migrant*", "refugee*")
))
```

```
manifestos <- readr::read_csv("../data/ireland_ge_2024_manifestos.csv")
```

```
manifestos_toks <- quanteda::tokens(
  manifestos$text,
  remove_punct = TRUE,
  remove_numbers = TRUE,
  remove_symbols = TRUE
)
```

Example: Dictionary Application

- Now we can apply the dictionary to the manifestos:

```
manifestos_imm <- quanteda::dfm(
  quanteda::tokens_lookup(manifestos_toks, dictionary = imm_dict)
)
```

```
manifestos_imm
```

```
Document-feature matrix of: 9 documents, 1 feature (0.00% sparse) and 0 docvars.
       features
docs    immigration
  text1          53
  text2          24
  text3          32
  text4          24
  text5          31
  text6          31
[ reached max_ndoc ... 3 more documents ]
```

Calculating Quantities of Interest

- Of course, the absolute number of matched terms is not, necessarily, informative.
- In the immigration focus example we can use the total number of matched terms M_i divided by the total number of words in the document N_i:

$$\text{immigration\_focus}_i = \frac{M_i}{N_i}$$

- If we were to try to scale the manifestos as pro- or anti- immigration (assuming we had a relevant dictionary), we could then try something like:

$$\text{immigration\_position}_i = \frac{M_i^{anti} - M_i^{pro}}{N_i}$$

- In other words, we would calculate an absolute proportional difference.

Scaling

- The previously described approach was used extensively in Manifesto Project.
- The problems, however, are:
  - Addition of irrelevant content shifts the scale toward zero.
  - Assumes the additional mentions increase emphasis in a linear scale
- One alternative (Laver & Garry, 2000):

$$\text{immigration\_position}_i = \frac{M_i^{anti} - M_i^{pro}}{M_i^{anti} + M_i^{pro}}$$

- Another alternative (Lowe, Benoit, Mikhaylov & Laver, 2011):

$$\text{immigration\_position}_i = \log \frac{M_i^{anti}}{M_i^{pro}}$$

Example: Dictionary Application

```r
immigration_focus <- cbind(
  manifestos,
  quanteda::convert(manifestos_imm, to = "data.frame")
) |>
  (\(df) transform(df, ntokens = quanteda::ntoken(manifestos_toks)))() |>
  (\(df) transform(df, rel_imm = immigration/ntokens))() |>
  _[, c("party", "immigration", "ntokens", "rel_imm")] |>
  (\(df) `[`(df, order(df$rel_imm, decreasing = TRUE),))()
```

```r
immigration_focus
```

```
      party immigration ntokens      rel_imm
text5    II          31    7295 0.0042494859
text7   PBP          26   11976 0.0021710087
text1    AO          53   27749 0.0019099787
text4    GR          24   29110 0.0008244589
text2    FF          24   33676 0.0007126737
text8    SD          36   58281 0.0006176970
text3    FG          32   52942 0.0006044350
text9    SF          28   48813 0.0005736177
text6   LAB          31   63107 0.0004912292
```

What to Do with Dictionary Results

- Describe the results.
- Scale the results (neg. vs pos., pro vs anti, left vs right, etc.).
- Could be used as features in downstream tasks:
  - Similarity measures (e.g. cosine)
  - ML-based classification
  - Topic modelling (seeded with keywords)
  - Prompt engineering for generative AI

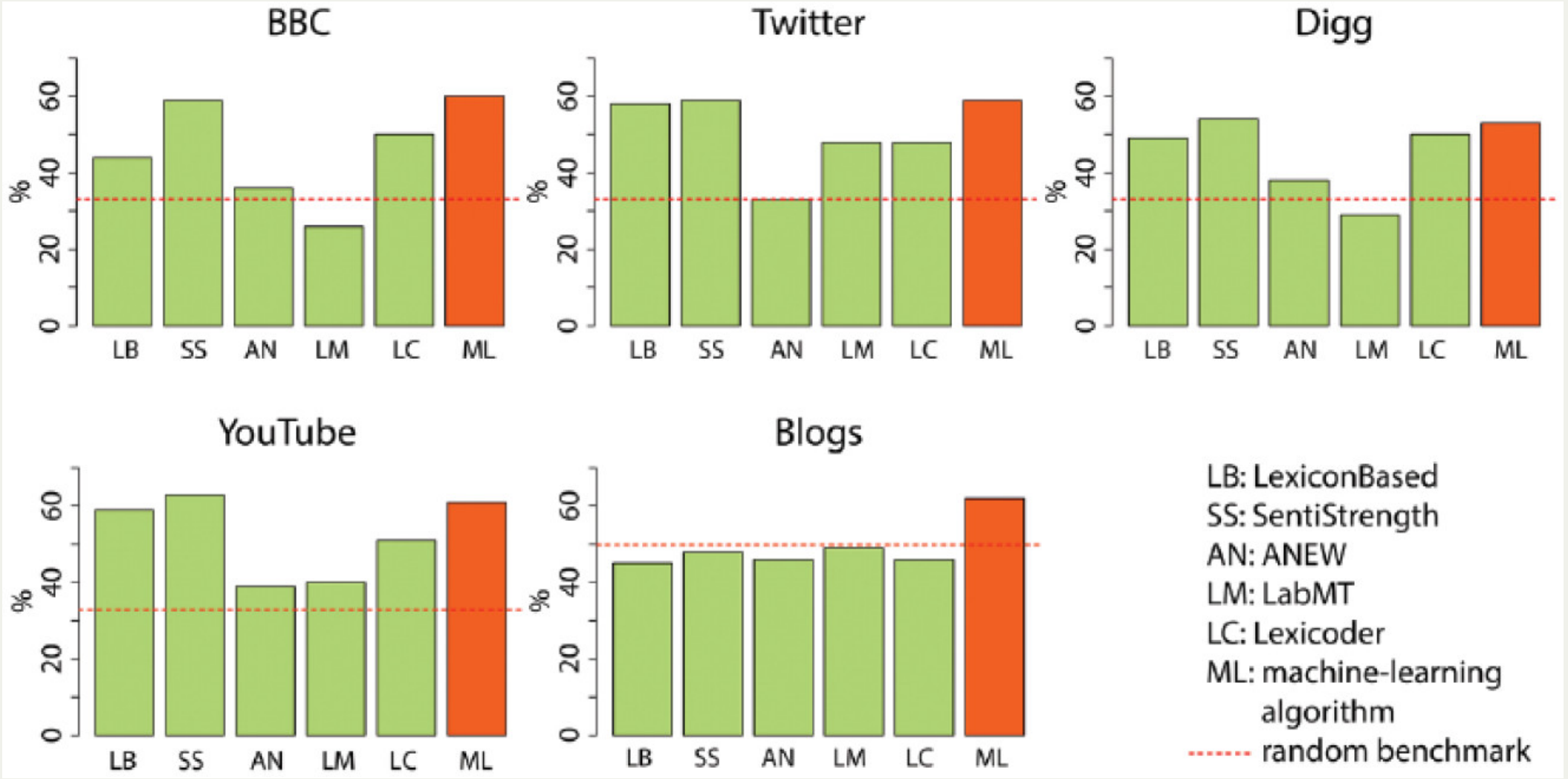
How to Build a Dictionary

1. Identify “extreme” texts with known positions.

- E.g. opposition leader and PM, one-star and five-star reviews, etc.

2. Search for differentially occurring words using word frequencies.
3. Examine these words in context to assess their sensitivity and specificity.
4. Examine inflected forms to see whether stemming or wildcarding is required.
5. Use these words (or stems/lemmas) for categories.

Dictionary Performance



(González-Bailón & Paltoglou, 2015)

Dictionary vs Machine Learning

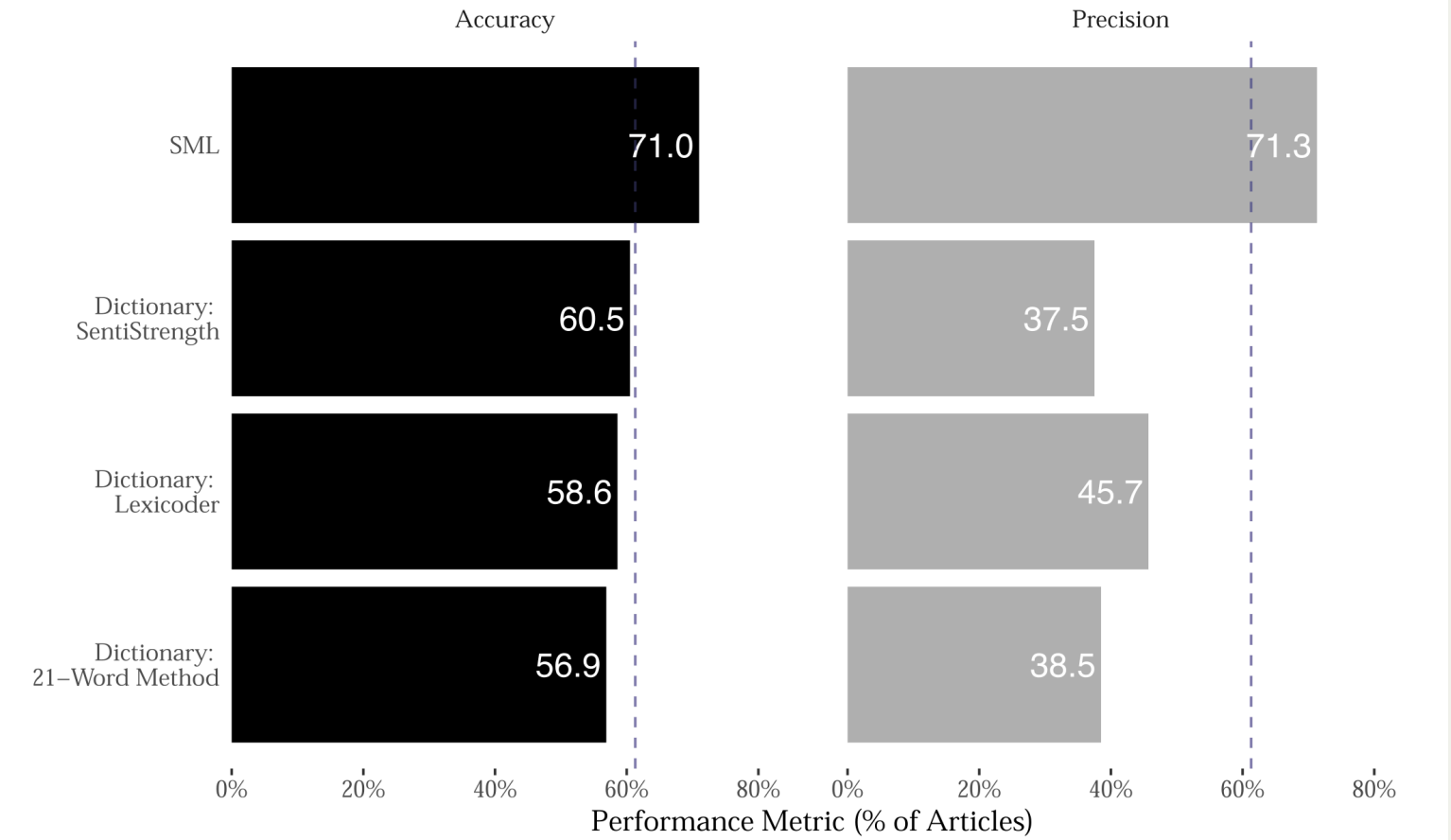


Figure 3. Performance of SML and Dictionary Classifiers—Accuracy and Precision.
Note: Accuracy (percent correctly classified) and precision (percent of positive predictions that are correct) for the ground truth dataset coded by ten CrowdFlower coders. The dashed vertical lines indicate the baseline 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).

(Barberá, Boydstun, Linn, McMahon & Nagler, 2021)

Next

- Tutorial: Dictionaries and text classification
- Next week: Supervised modelling of text