Lecture 10: Similarity Metrics and Supervised Learning for Text

Jack Blumenau

The London School of Economics and Political Science

Today’s lecture

Motivating Example

How similar are these two modules?

```
[1] "Causal Inference (PUBL0050)"
```

```
[1] "This course provides an introduction to statistical methods used for causal inference in the social
sciences. We will be concerned with understanding how and when it is possible to make causal claims in empirical
research. In particular, we will focus on understanding which assumptions are necessary for giving research a
causal interpretation, and on learning a range of approaches that can be used..."
```

```
[1] "Quantitative Text Analysis for Social Science (PUBL0099)"
```

```
[1] "Growth of text data in recent years, and the development of a set of sophisticated tools for analysing that
data, offers important opportunities for social scientists to study questions that were previously amenable to
only qualitative analyses.\n\nThis module will allow students to take advantage of these opportunities by
providing them with an understanding of, and ability to apply, tools of quantitative text analysis..."
```

We will use data from the universe of modules taught at UCL to evaluate the similarity between these courses.

Module catalogue

Module Catalogue

UCL's Module Catalogue is a resource for both staff and students, and provides summary information on all of the modules running at the University during the academic year 2022/23.

Important information

The catalogue has been updated with key information about the modules that will run during the 2022/23 academic session. Current and prospective students can browse through the catalogue to consider possible module choices for the coming year.

Please note, information in the catalogue is subject to change as teaching and assessment arrangements for 2022/23 may need to be adjusted in line with the Teaching and Assessment Operating Model.

Centrally managed exam durations will be provided on students' individual exam timetables. Arrangements for locally managed exams and tests will be confirmed by the teaching department for the module.

f you are a current student, the module catalogue will help you to find information about modules within your lepartment and across UCL. The module catalogue may also be useful if you are a prospective student and want o know more about the modules available if you take a particular course.

'ou can search for modules by department, title, keywords, codes and/or credit value

Useful links

- Disclaimer
- Glossary of terminology
- Modules not included in the catal
- Student module selection
- Your UCL education in the 2022/2 academic year

Sustainability

For a list of modules related to climate change as well as social and environm sustainability at UCL, type **'climate'** or **'sustainability'** into the search. Visit Sustainable UCL for information on ext curricular activity on sustainability.

Similarity

Vector Space Model

- We previously represented our text data as a document-feature matrix
  - Rows: Documents
  - Columns: Features
- Each document is therefore described by a **vector** of word counts
- This representation allows us to measure several important properties of our documents

Vectors notation

We denote a vector representation of a document using a **bold** letter:

$$\mathbf{a} = \{a_1, a_2, \ldots, a_J\}$$

where a_1 is the number of times feature 1 appears in the document, a_2 is the number of times feature 2 appears in the document, and so on.

Similarity

- **Idea:** Each document can be represented by a vector of (weighted) feature counts, and that these vectors can be evaluated using **similarity** metrics
- A document's vector is simply (for now) it's row in the document-feature matrix
- **Key question:** how do we measure distance or similarity between the vector representation of two (or more) different documents?

Similarity

There are many different metrics we might use to capture similarity/difference between texts:

1. Edit distances
2. Inner product
3. Euclidean distance
4. Cosine similarity

The choice of metric comes down to an assumption about which kinds of differences are most important to consider when comparing documents.

Edit Distance

- Edit distances measure the similarity/difference between text strings
- A commonly used edit distance is the **Levenshtein distance**
- Measures the minimal number of operations (replacing, inserting, or deleting) required to transform one string into another
- Example: the Levenshtein distance between “kitten” and “sitting” is 3
  - **k**itten ➡️ **s**itten (substitute “k” for “s”)
  - sitt**e**n ➡️ sitt**i**n (substitute “e” for “i”)
  - sittin ➡️ sittin**g** (insert “g” at the end)
- In `r`:

```r
x <- c("kitten", "sitting")

adist(x)
```

```
     [,1] [,2]
[1,]    0    3
[2,]    3    0
```

- Generally not used in large scale applications because computationally burdensome to implement on long texts

Inner Product

Inner product

The inner product, or “dot” product, between two vectors is the sum of the element-wise multiplication of the vectors:

$$
\begin{aligned}
\mathbf{a} \cdot \mathbf{b} &= \mathbf{a}^T \mathbf{b} \\
&= a_1 b_1 + a_2 b_2 + \ldots + a_J b_J
\end{aligned}
$$

NB: dot product is a *scalar*

NB: When the vectors are *dichotomized* document-feature matrices (only 0s and 1s), then the inner product gives the number of features that the two documents share in common.

Example

Imagine three documents with a six-word vocabulary:

	causal	estimate	identification	text	document	feature
Document a	2	3	3	0	0	1
Document b	2/4	0	0	3/6	2/4	3/6
Document c	1	2	1	1	0	1

Euclidean Distance

Euclidean Distance

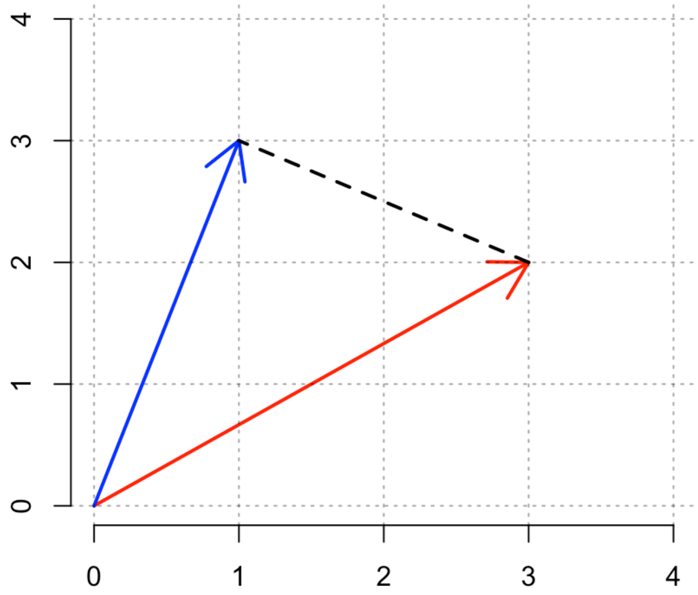
The Euclidean Distance between two document vectors, $\mathbf{a}$ and $\mathbf{b}$, is given by:

$$
\begin{aligned}
d(\mathbf{a}, \mathbf{b}) &= \sqrt{\sum_{j=1}^{J} (a_j - b_j)^2} \\
&= ||\mathbf{a} - \mathbf{b}||
\end{aligned}
$$

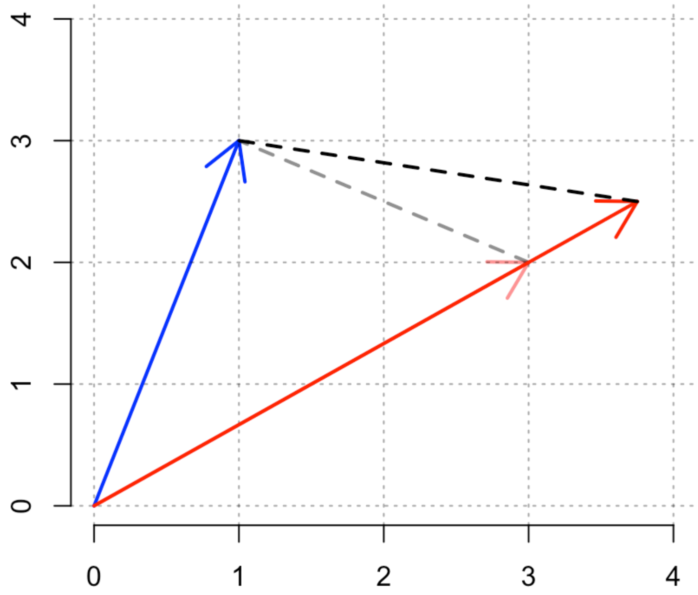
Where J is the total number of features in the dfm.

- The Euclidean distance is based on the Pythagorean theorem
- Similar problem to the inner product: sensitive to document length

Euclidean Distance Illustration



Euclidean Distance Illustration



Cosine Similarity

- Measures of document similarity should not be sensitive to the number of words in each of the documents
  - We don't want long documents to be "more similar" than shorter documents just as a function of length
- A natural way to adapt the inner product measure is to normalise by document length, which we do by calculating the **magnitude** of the document vectors
- **Cosine similarity** is a measure of similarity that is based on the *normalized* inner product of two vectors
- It can be interpreted as…
  - …a normalized version of the inner product or Euclidean distance
  - …the cosine of the *angle* between the two vectors

Cosine Similarity

Cosine similarity

The cosine similarity ($cos(\theta)$) between two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||}$$

where θ is the angle between the two vectors and $||\mathbf{a}||$ and $||\mathbf{b}||$ are the *magnitudes* of the vectors $\mathbf{a}$ and $\mathbf{b}$, respectively.

Vector Magnitude (or “length”)

The magnitude of a vector (also known as the “length”) is the square-root of the inner product of the vector with itself:

$$\begin{aligned} ||\mathbf{a}|| &= \sqrt{\mathbf{a} \cdot \mathbf{a}} \\ &= \sqrt{a_1^2 + a_2^2 + \ldots + a_J^2} \end{aligned}$$

Interpretation

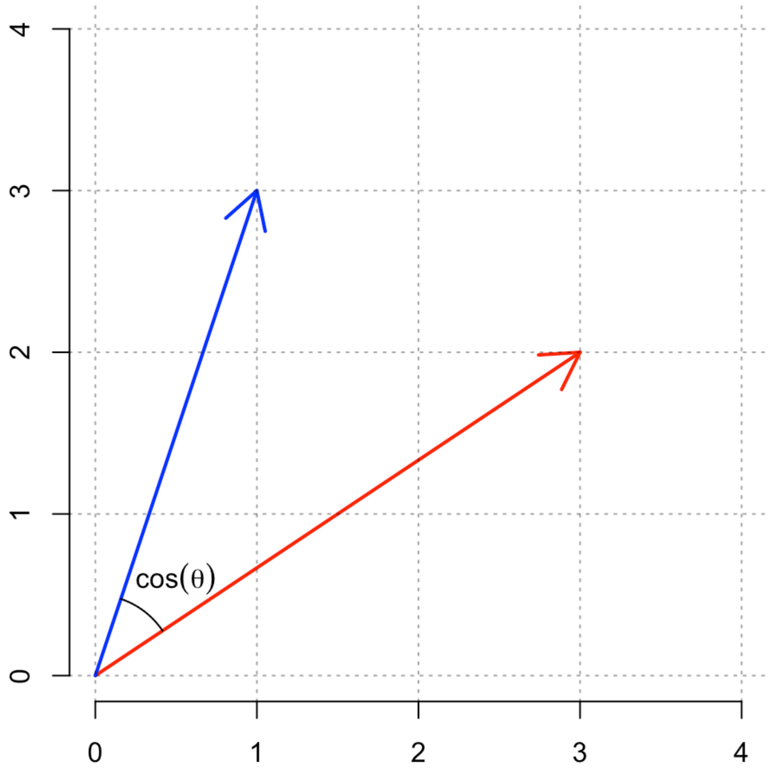
The value of cosine similarity ranges from -1 to 1

- A value of 1 indicates that the vectors are identical
- A value of 0 indicates that the vectors are orthogonal (i.e., not similar at all)
- A value of -1 indicating that the vectors are diametrically opposed.

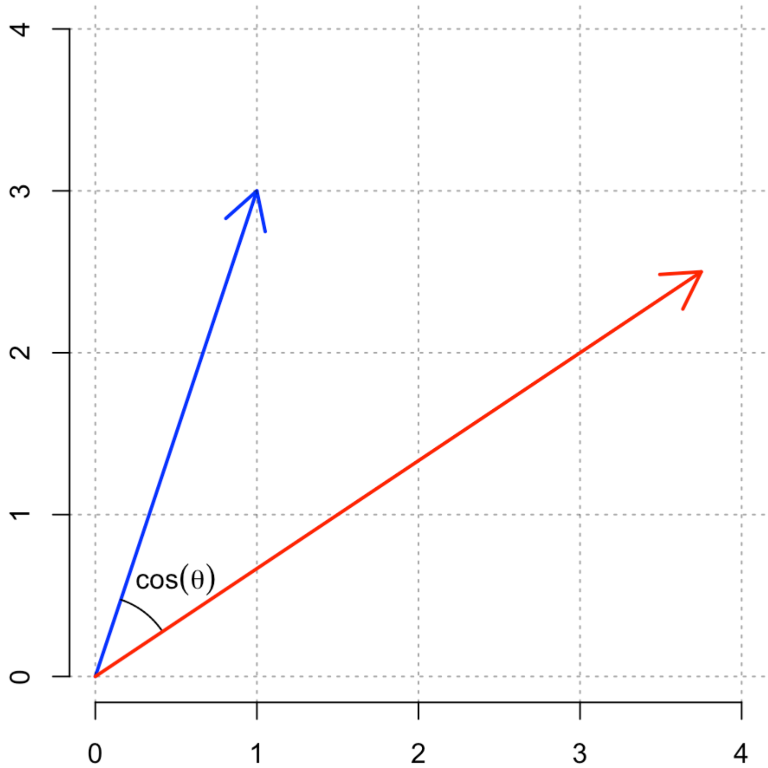
Thus, the closer the value is to 1, the more similar the vectors are.

Calculated for vectors of word *counts* (or any positively-valued vectors), the cosine similarity ranges from 0 to 1.

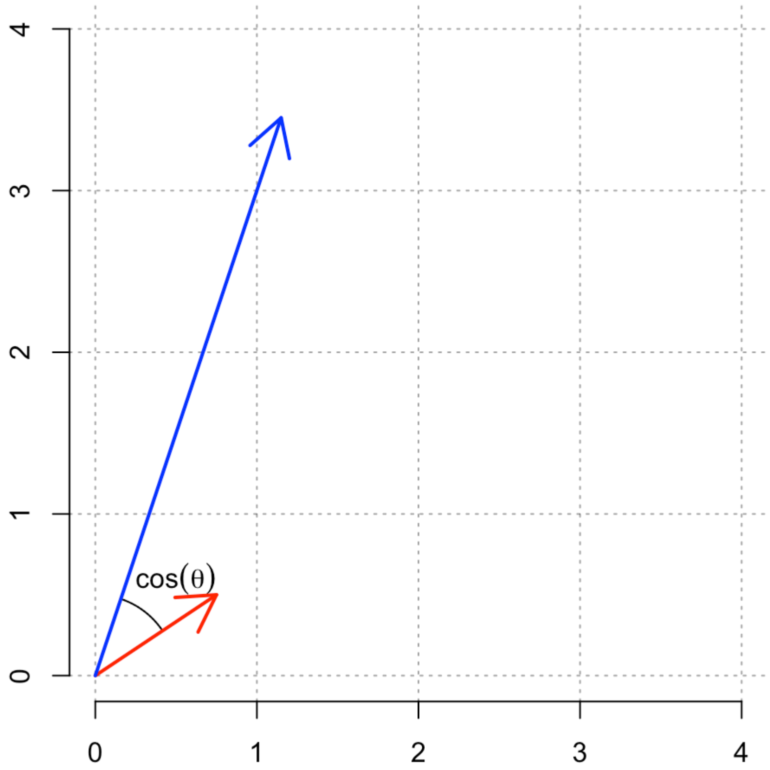
Cosine Similarity Illustration



Cosine Similarity Illustration



Cosine Similarity Illustration



Module Catalogue Data

```
str(modules)
```

```
tibble [6,248 × 10] (S3: tbl_df/tbl/data.frame)
 $ teaching_department   : chr [1:6248] "Greek and Latin" "Greek and Latin" "Bartlett School of Sustainable C
 $ level                 : num [1:6248] 5 4 7 5 7 7 4 7 7 7 ...
 $ intended_teaching_term: chr [1:6248] "Term 1|Term 2" "Term 1" "Term 1" "Term 2" ...
 $ credit_value          : chr [1:6248] "15" "15" "15" "30" ...
 $ mode                  : chr [1:6248] "" "" "" "" ...
 $ subject               : chr [1:6248] "Ancient Greek|Ancient Languages and Cultures|Classics" "Ancient Gree
 $ keywords              : chr [1:6248] "ANCIENT GREEK|LANGUAGE" "ANCIENT GREEK|LANGUAGE" "Infrastructure fin
 $ title                 : chr [1:6248] "Advanced Greek A (GREK0009)" "Greek for Beginners A (GREK0002)" "Inf
 $ module_description    : chr [1:6248] "Teaching Delivery: This module is taught in 20 bi-weekly lectures an
 $ code                  : chr [1:6248] "GREK0009" "GREK0002" "BCPM0016" "BARC0135" ...
```

```
modules$module_description[modules$code == "PUBL0099"]
```

```
[1] "Growth of text data in recent years, and the development of a set of sophisticated tools for analysing t
```

```
modules$module_description[modules$code == "PUBL0050"]
```

```
[1] "This course provides an introduction to statistical methods used for causal inference in the social scie
```

Question: Which other modules at UCL are most similar to these two modules?

Cosine Similarity – Application

PUBL0050

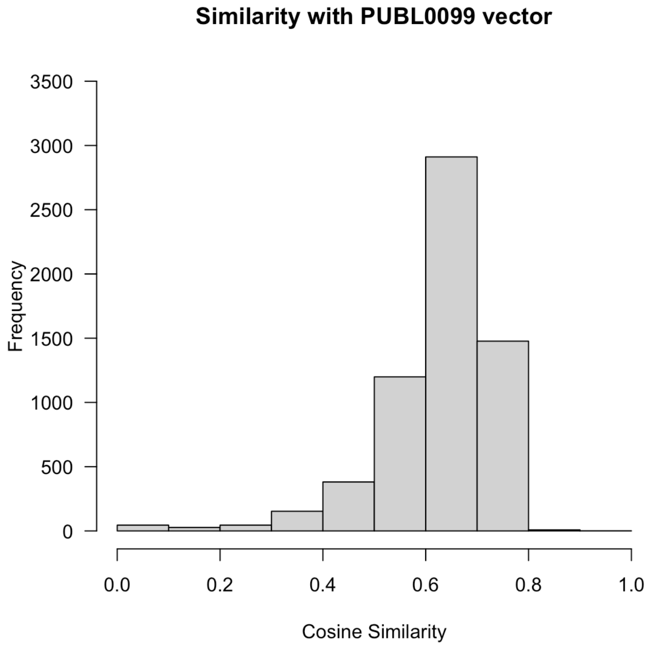
```
# Create a corpus object from module catalogue data
modules_corpus <- corpus(modules,
                         text_field = "module_description",
                         docid_field = "code")

# Convert modules data into a dfm
modules_dfm <- modules_corpus %>%
                tokens() %>%
                dfm()

# Calculate the cosine similarity between PUBL0050 and all other modules
cosine_sim_50 <- textstat_simil(x = modules_dfm,
                                y = modules_dfm[modules$code == "PUBL0050",],
                                method = "cosine")

head(cosine_sim_50)
```

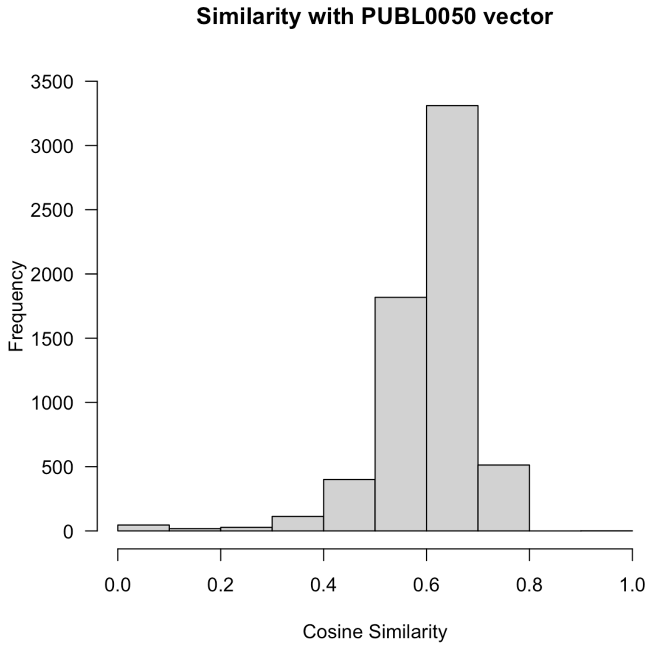
```
          PUBL0050
GREK0009 0.6801510
GREK0002 0.6209725
BCPM0016 0.5782462
BARC0135 0.5060876
BCPM0036 0.4374233
BIDI0002 0.6731816
```

PUBL0099

```
# Calculate the cosine similarity between PUBL0099 and all other modules
cosine_sim_99 <- textstat_simil(x = modules_dfm,
                                y = modules_dfm[modules$code == "PUBL0099",],
                                method = "cosine")
```

Cosine Similarity – Application





Cosine Similarity – Application

Which modules are most similar to PUBL0050?

```r
# Create a new variable in original data frame
modules$cosine_sim_50 <- as.numeric(cosine_sim_50)

# Arrange the data.frame in order of similarity an
modules %>%
  arrange(-cosine_sim_50) %>%
  select(title)
```

```
# A tibble: 6,248 × 1
   title
   <chr>
 1 Causal Inference (PUBL0050)
 2 Research Methods and Skills (ANTH0104)
 3 Regression Modelling (IEHC0050)
 4 Selected Topics in Statistics (STAT0017)
 5 Dissertation - MSc CPIPP (PHAY0053)
 6 Advanced Photonics Devices (ELEC0109)
 7 User-Centred Data Visualization (PSYC0102)
 8 Introduction to Assessment (MDSC0002)
 9 Quantitative Methods and Mathematical Thinking
10 Core Principles of Mental Health Research (PSBS
# ℹ 6,238 more rows
```

Which modules are most similar to PUBL0099?

```r
# Create a new variable in original data frame
modules$cosine_sim_99 <- as.numeric(cosine_sim_99)

# Arrange the data.frame in order of similarity an
modules %>%
  arrange(-cosine_sim_99) %>%
  select(title)
```

```
# A tibble: 6,248 × 1
   title
   <chr>
 1 Quantitative Text Analysis for Social Science (
 2 Archaeological Glass and Glazes (ARCL0099)
 3 User-Centred Data Visualization (PSYC0102)
 4 Understanding and Analysing Data (SESS0006)
 5 Understanding and Analysing Data (SEES0107)
 6 Data Analysis (POLS0010)
 7 Laboratory and Instrumental Skills in Archaeolo
 8 The Anthropology of Violent Aftermaths (ANTH013
 9 Fashion Cultures (LITC0044)
10 Anthropology of Politics, Violence and Crime (A
```

Misleading Word Counts

Why do we recover so many strange matches for our PUBL0050 and PUBL0099 documents?

Let's compare the most common features of the following four modules:

PUBL0099 – Quantitative Text Analysis for Social Science

```r
topfeatures(modules_dfm[modules$code=="PUBL0099",]
```

```
   ,  and   of   to    .  the   in text
  21   12   12   11   10   10    8    8
```

PUBL0050 – Causal Inference

```r
topfeatures(modules_dfm[modules$code=="PUBL0050",]
```

```
  .  in   ,  to and the  of   ;
 14  11  11  10   9   9   8   8
```

ELEC0109 – Advanced Photonics Devices

```r
topfeatures(modules_dfm[modules$code=="ELEC0109",]
```

```
and   , of the   ;   .  to  in
104 100  77  77  62  55  51  48
```

ARCL0099 – Archaeological Glass and Glazes

```r
topfeatures(modules_dfm[modules$code=="ARCL0099",]
```

```
   ,  the   of  and   to    .   in this
  27   24   23   20   17   16    9    6
```

Feature selection matters! Similarities here are being driven by substantively unimportant words.

Weighted Vectors

- The bag-of-words representation characterises documents according to the raw counts of each word
- The critical problem with using raw term frequency is that all terms are considered equally important when it comes to assessing similarity
- One way of avoiding this problem is to **weight** the vectors of word counts in ways that make our text representations more informative
- There are several strategies for weighting the word vectors that represent our documents, the most common of which is tf-idf weighting

Tf-idf intuition

- Tf-idf stands for “term-frequency-inverse-document-frequency”
- Tf-idf weighting can improve our representations of documents because it assigns higher weights to…
  - … words that are *common* in a given document (“term-frequency”) and
  - … words that are *rare* in the corpus as a whole (“*inverse*-document-frequency”)
- Down-weighed words include…
  - …stop words (e.g. and, if, the, but, etc) and also…
  - … terms that are domain-specific but used frequently across documents (e.g. module, class, assessment, exam)
- Up-weighted terms are therefore those words that are more *distinctive* and thus are more useful for characterising a given text

TF-idf

Term-frequency-inverse-document-frequency (tf-idf)

The *tf-idf* weighting scheme assigns to feature j a weight in document i according to:

$$\begin{aligned}\text{tf-idf}_{i,j} &= W_{i,j} \times idf_j \\ &= W_{i,j} \times log(\frac{N}{df_j})\end{aligned}$$

- $W_{i,j}$ is the number of times feature j appears in document i
- df_j is the number of documents in the corpus that contain feature j
- N is the total number of documents

NB: tf-idf is specific to a feature in a document

Implications

$\text{tf-idf}_{i,j}$ will be…

1. …highest when feature j occurs many times in a small number of documents
2. …lower when feature j occurs few times in a document, or occurs in many documents
3. …lowest when feature j occurs in virtually all documents

Tf-idf – Application

```r
# Convert modules data into a dfm *with tf-idf wieghts*
modules_dfm_tfidf <- modules_corpus %>%
                        tokens() %>%
                        dfm() %>%
                        dfm_tfidf()

modules_dfm_tfidf
```

```
Document-feature matrix of: 6,248 documents, 35,483 features (99.68% sparse) and 8 docvars.
          features
docs        teaching delivery       :      this    module        is   taught
  GREK0009 0.8071821 1.083091 2.74080 0.3076292 0.5888072 0.3026049 1.791841
  GREK0002 1.6143641 1.083091 1.64448 0.0769073 0.3680045 0.1513024 1.791841
  BCPM0016 0         1.083091 0.54816 0.2307219 0.2208027 0.1513024 0
  BARC0135 0         0        0.82224 0.0769073 0         0.3026049 0
  BCPM0036 0         0        1.09632 0.0769073 0.0736009 0         0
  BIDI0002 0         0        0.27408 0.2307219 0.2944036 0.1513024 0
          features
docs               in       20 bi-weekly
  GREK0009 0.43350179 1.851258  2.453318
  GREK0002 0.07881851 1.851258  2.453318
  BCPM0016 0.03940925 0         0
  BARC0135 0          0         0
  BCPM0036 0.03940925 0         0
  BIDI0002 0.15763701 0         0
[ reached max_ndoc ... 6,242 more documents, reached max_nfeat ... 35,473 more features ]
```

Tf-idf – Application

What are the features with the highest tf-idf scores for our four modules?

PUBL0099 – Quantitative Text Analysis for Social Science

```
topfeatures(modules_dfm_tfidf[modules$code=="PUBL0
```

```
        text     digitized quantitative        count
   11.992607      7.591482     5.431962      5.36359
     collect         texts
    4.049778      4.030291
```

PUBL0050 – Causal Inference

```
topfeatures(modules_dfm_tfidf[modules$code=="PUBL0
```

```
      causal     causality   regression
    7.093132      5.183242     5.065593      4.63449
quantitative        claims
    4.073971      3.979122
```

ELEC0109 – Advanced Photonics Devices

```
topfeatures(modules_dfm_tfidf[modules$code=="ELEC0
```

```
            ;       optical         laser        l
     35.91729      35.64063      31.79536     28.
     photonic        liquid       devices
     25.16167      24.78914      24.23796
```

ARCL0099 – Archaeological Glass and Glazes

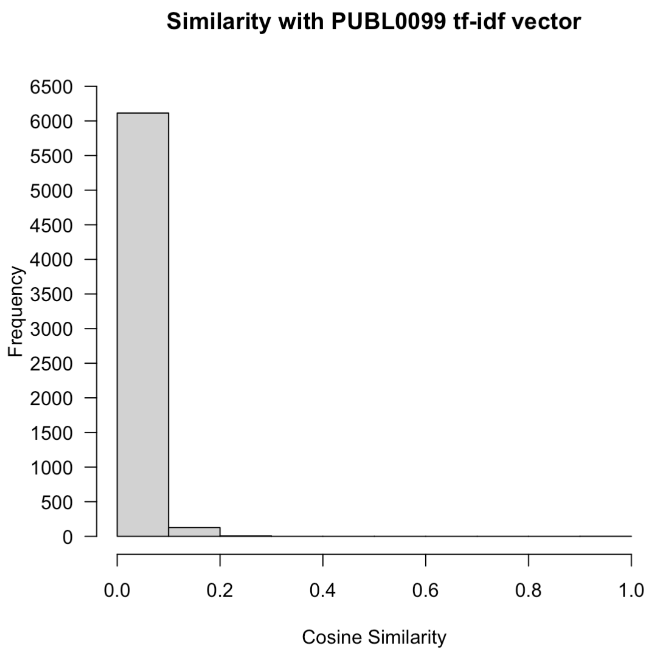
```
topfeatures(modules_dfm_tfidf[modules$code=="ARCL0
```

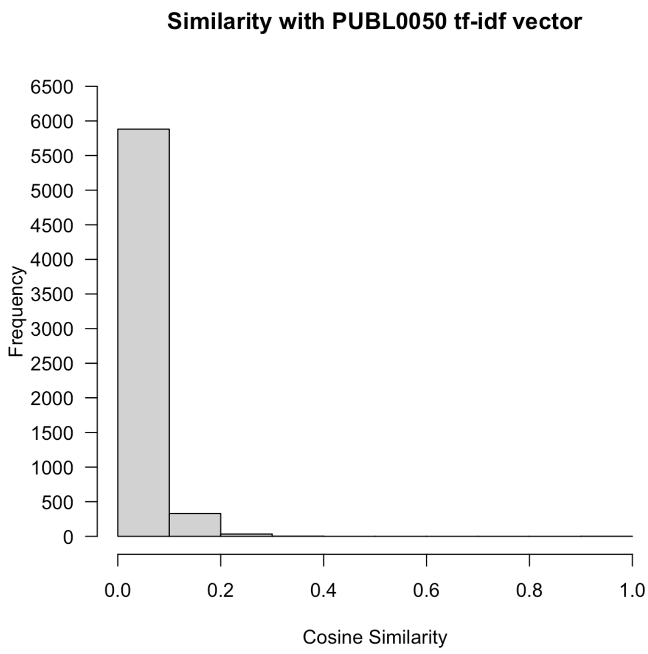
```
        glass        glazes      pigments
    14.753215      7.591482      6.989422      6.6
chronological     siliceous    ornamental
     4.285057      3.795741      3.795741
```

Tf-idf cosine similarity

```
# Calculate the cosine similarity between PUBL0050 and all other modules
cosine_sim_tfidf_50 <- textstat_simil(x = modules_dfm_tfidf,
                             y = modules_dfm_tfidf[modules$code == "PUBL0050",],
                             method = "cosine")

# Calculate the cosine similarity between PUBL0099 and all other modules
cosine_sim_tfidf_99 <- textstat_simil(x = modules_dfm_tfidf,
                             y = modules_dfm_tfidf[modules$code == "PUBL0099",],
                             method = "cosine")
```

Tf-idf – Application





Cosine Similarity – Application

Which modules are most similar to PUBL0050?

```
# Create a new variable in original data frame
modules$cosine_sim_tfidf_50 <- as.numeric(cosine_s

# Arrange the data.frame in order of similarity an
modules %>%
  arrange(-cosine_sim_tfidf_50) %>%
  select(title)
```

```
# A tibble: 6,248 × 1
   title
   <chr>
 1 Causal Inference (PUBL0050)
 2 Causal Analysis in Data Science (POLS0012)
 3 Advanced Quantitative Methods (PHDE0084)
 4 Quantitative Data Analysis (POLS0083)
 5 Advanced Statistics for Records Research (CHME0
 6 Understanding and Analysing Data (SESS0006)
 7 Understanding and Analysing Data (SEES0107)
 8 Quantitative and Qualitative Research Methods 1
 9 Statistics for Health Economics (STAT0039)
10 Introduction to Statistics for Social Research
# ℹ 6,238 more rows
```

Which modules are most similar to PUBL0099?

```
# Create a new variable in original data frame
modules$cosine_sim_tfidf_99 <- as.numeric(cosine_s

# Arrange the data.frame in order of similarity an
modules %>%
  arrange(-cosine_sim_tfidf_99) %>%
  select(title)
```

```
# A tibble: 6,248 × 1
   title
   <chr>
 1 Quantitative Text Analysis for Social Science (
 2 Data Science for Crime Scientists (SECU0050)
 3 Understanding and Analysing Data (SESS0006)
 4 Understanding and Analysing Data (SEES0107)
 5 Data Analysis (POLS0010)
 6 Quantitative Data Analysis (POLS0083)
 7 Literary Linguistics A (ENGL0042)
 8 Analysing Research Data (IOEF0026)
 9 Middle Bronze Age to the Iron Age in the Near E
10 Research Methods - Quantitative (CENG0045)
```

Tf-idf Does Not Solve All Problems

Consider these two sentences:

- "Quantitative text analysis is very successful."
- "Natural language processing is tremendously effective."

Represented as a DFM:

	quantitative	text	analysis	very	successful	natural	language	processing
D1	1	1	1	1	1	0	0	0
D2	0	0	0	0	0	1	1	1

The cosine similarity between these vectors is:

$$cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||} = 0$$

No dfm weighting scheme can address the core problem: *the sentences are formed of non-overlapping sets of words*.

We will see one powerful alternative to this problem when we consider word embedding

Cosine Similarity Example

Does public opinion affect political speech? (Hager and Hilbig, 2020)

Does learning about the public's attitudes on a political issue change how much attention politicians pay to that issue in their public statements?

Set up:

- Politicians in Germany have historically received public opinion research on citizens' attitudes
- Release of the polling data is exogenously determined, providing causal identification (via a regression-discontinuity design)
- Strategy: Measure the linguistic (cosine) similarity between reports summarising public opinion and political speeches

Cosine Similarity Example

Table 2 Effects on Cosine Similarity

	Cosine Similarity	
	(1)	(2)
Exposure	0.0137**	0.0128**
	(0.0066)	(0.0057)
Covariates	No	Yes
Observations	5,684	5,684
Mean of DV	0.1263	
SD of DV	0.0976	
Effect size in SD	0.1413	0.1319

Note: The table reports results from a local linear regression around the release of the opinion reports (optimal bandwidth of 22 days; Equation 1). The outcome is the cosine similarity between reports and speeches. The sample is limited to pairs where both speech document and opinion report address the same topic. In Model 2, all covariates reported in Table 1 are included. Standard errors in parentheses are clustered by speech document and by opinion report.
*p<.1; **p<.05; ***p<.01.

Implication: Public statements of politicians move closer to summaries of public opinion after publication of polling research

Difference

Detecting discriminating words

Sometimes we want to *characterise differences* between documents, not just measuring the similarity between them.

We want to find a set of words that conveys the **distinct** content between documents.

We might be interested in, for example, how language use differs between…

1. …politicians on the left and the right (Diermeier et. al., 2012)
2. …male and female voters (Cunha, et. al., 2014)
3. …blog posts written by people in different geographic regions (Eisenstein et. al., 2010)

Identifying discriminating words between groups is useful because these words tend to help us characterise the type of language/arguments/linguistic frames that a group employs.

Word clouds

The high-dimensional nature of natural language means that often the best methods for detecting discriminating words are those that allow us to **visualise** differences between groups.

A very common method for visualising corpus- or group-wide word use is via a **word cloud**.

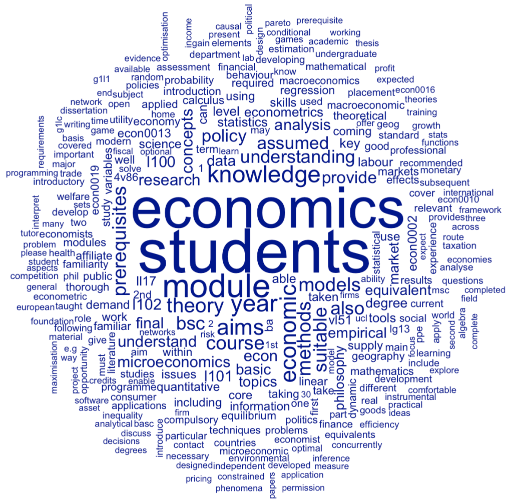
- A word cloud is a visual representation of the frequency and importance of words in a given text.
- The size of each word in the cloud reflects its frequency or importance within the text.
- The layout of the words in the cloud is usually **random**, but it is possible to arrange terms such that their placement reflects some variation of interest

We will use word clouds to explore the differences between Economics and Political Science modules.

Word clouds – Application

```r
# Load library for plotting
library(quanteda.textplots)

# Remove stopwords
modules_dfm <- modules_dfm %>% dfm_remove(stopwords("en"))

# Subset the modules_dfm object to only modules in PS or Econ
ps_dfm <- modules_dfm[docvars(modules_dfm)$teaching_department == "Political Science",]
econ_dfm <- modules_dfm[docvars(modules_dfm)$teaching_department == "Economics",]

# Create word clouds with top 300 features in each dfm
textplot_wordcloud(ps_dfm, max_words = 300)
textplot_wordcloud(econ_dfm, max_words = 300)
```

Word clouds – Application



Although there are some differences, many words are common across both sets of document.

Word clouds – Application

```r
library(quanteda.textplots)

# Create a corpus object from module catalogue data
modules_corpus <- corpus(modules,
                         text_field = "module_description",
                         docid_field = "code")

# Convert modules data into a difm
ps_econ_dfm_tf_idf <- modules_corpus %>%
                tokens(remove_punct = T) %>%
                dfm() %>%
                dfm_tfidf()
```

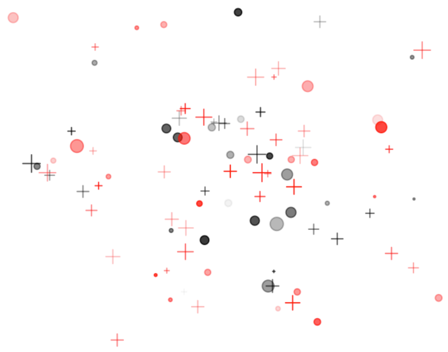
Word clouds – Application

Even with tf-idf weighting, it is hard to identify many of the distinguishing words.

The Problem with Word Clouds

Humans can only visualise a limited number of dimensions:

- Width (i.e. x-axis position)
- Height (i.e. y-axis position)
- Depth (i.e. z-axis position, hard for most people)
- Colour
- Shape
- Size
- Opacity



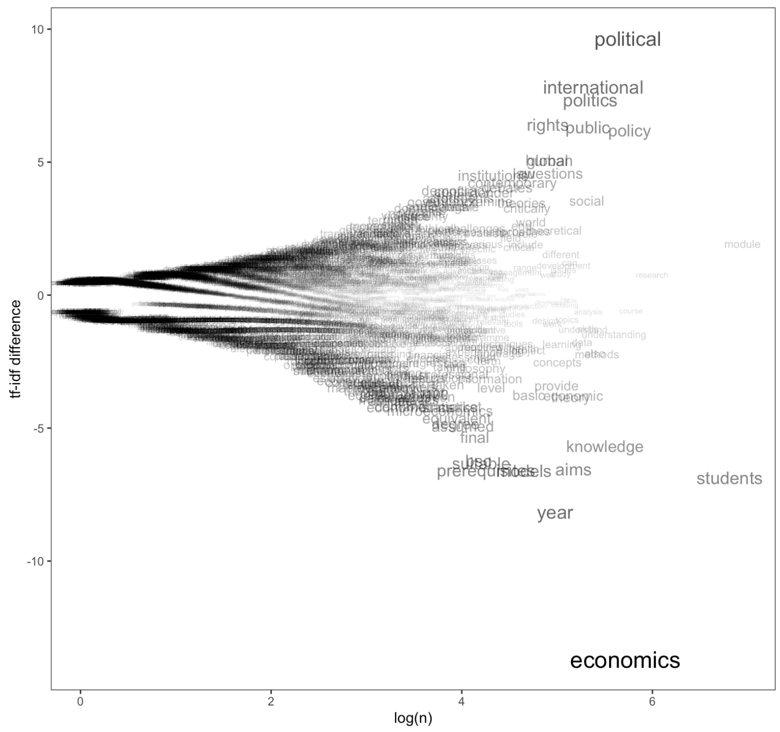
Core problem of word clouds: they do not take full advantage of the dimensional

Discriminating Words

An alternative approach is to directly visualise the **difference** in word use across groups.

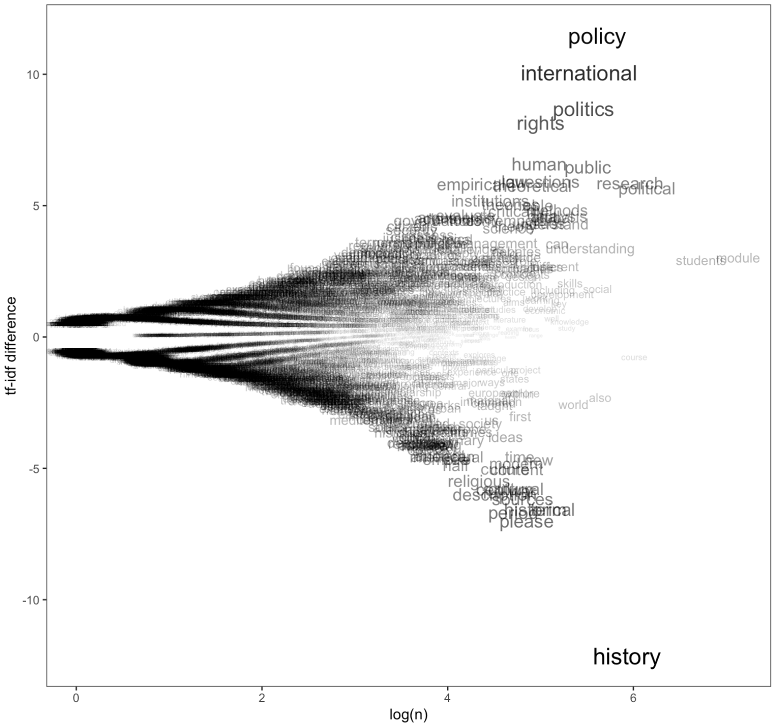
We need a metric to calculate such differences at the individual feature level. One such approach is to use the difference in tf-idf scores across groups (Munroe, et. al, 2008).

Discriminating words – Political Science v Economics



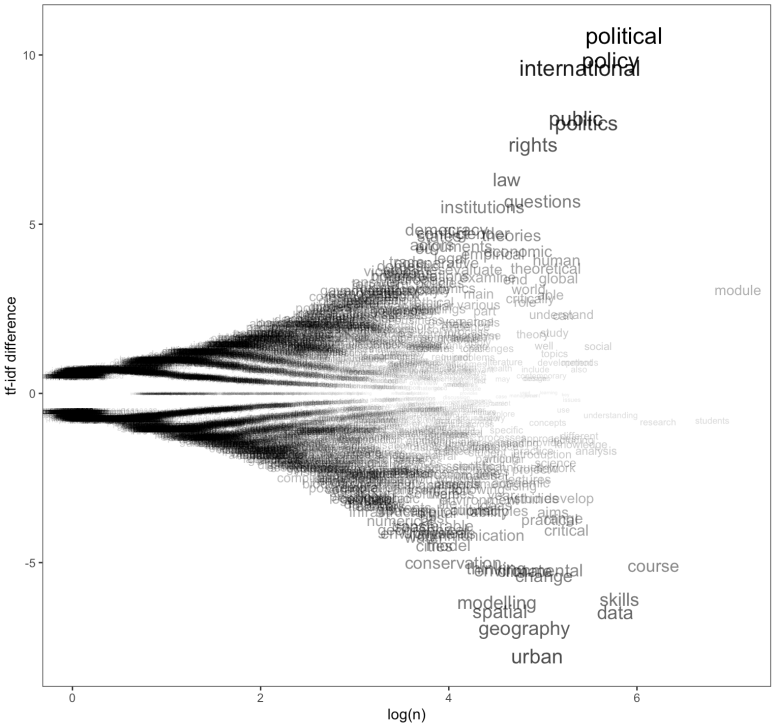
Political Science	Economics
political	economics
international	year
politics	students
rights	models
public	prerequisites
policy	aims
human	suitable
global	bsc
law	knowledge
questions	final

Discriminating words – Political Science v History



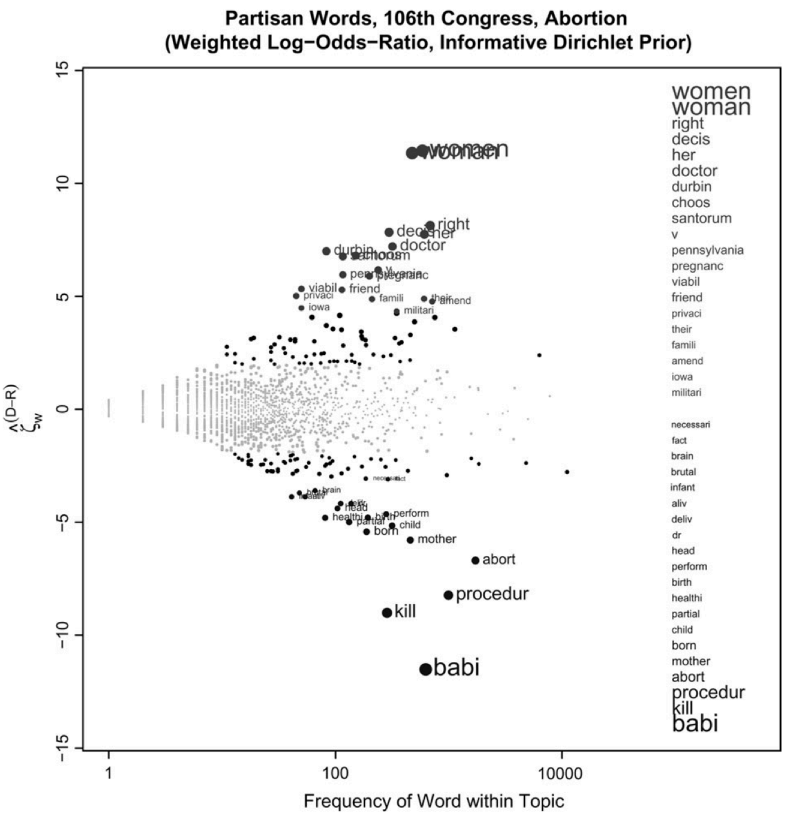
Political Science	History
policy	history
international	please
politics	period
rights	term
human	historical
public	sources
questions	description
law	full
research	war
empirical	century
theoretical	cultural

Discriminating words – Political Science v Geography



Political Science	Geography
political	urban
policy	geography
international	data
public	spatial
politics	modelling
rights	skills
law	change
questions	climate
institutions	environmental
democracy	thinking

Example: Fightin’ words



Example: Fightin’ words

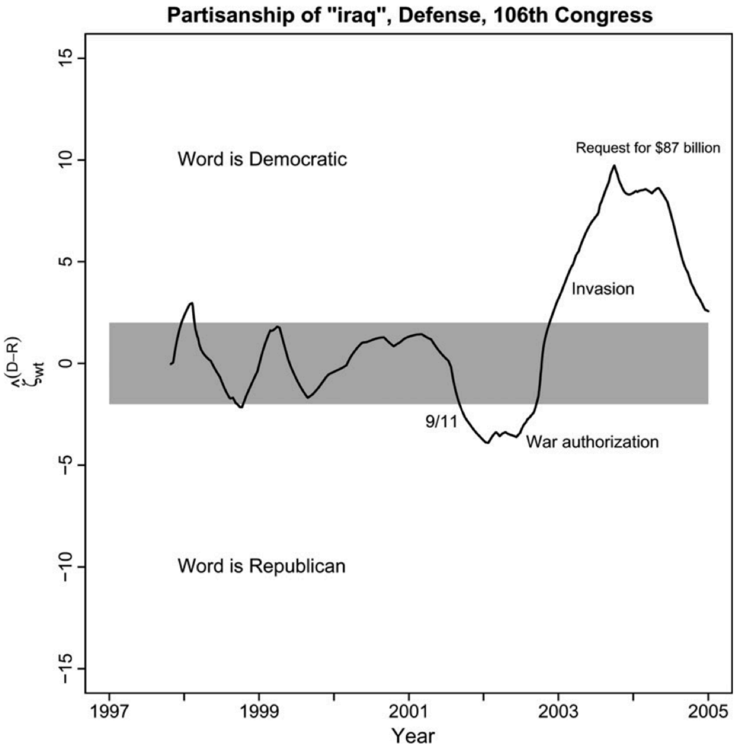


Fig. 9 Dynamic partisanship of “iraq” in the context of defense.

Break

Supervised Learning for Text

Motivation - Is this a curry?

Motivation

What is a curry?

Oxford English Dictionary:

> “A preparation of meat, fish, fruit, or vegetables, cooked with a quantity of bruised spices and turmeric, and used as a relish or flavouring, esp. for dishes composed of or served with rice. Hence, a curry = a dish or stew (of rice, meat, etc.) flavoured with this preparation (or with curry-powder).”

Motivation

- If a curry can be defined by the spices a dish contains, then we ought to be able to predict whether a recipe is a curry from ingredients listed in recipes
- We will evaluate the probability that #TheStew is a curry by training a curry classifier on a set of recipes
- We will use data on 9384 recipes from the BBC recipe archive
- This data includes information on
  - Recipe names
  - Recipe ingredients
  - Recipe instructions

Motivation

Our data includes information on each recipe:

```r
recipes$recipe_name[1]
```

```
[1] "Mustard and thyme crusted rib-eye of beef "
```

```r
recipes$ingredients[1]
```

```
[1] "2.25kg/5lb rib-eye of beef, boned and rolled 450ml/Â¾ pint red wine 150ml/Â¼ pint red wine vinegar 1 tbsp
sugar 1 tsp ground allspice 2 bay leaves 1 tbsp chopped fresh thyme 2 tbsp black peppercorns, crushed 2 tbsp
English or Dijon mustard"
```

```r
recipes$directions[1]
```

```
[1] "Place the rib-eye of beef into a large non-metallic dish. In a jug, mix together the red wine, vinegar,
sugar, allspice, bay leaf and half of the thyme until well combined. Pour the mixture over the beef, turning to
coat the joint evenly in the liquid. Cover the dish loosely with cling film and set aside to marinate in the
fridge for at least four hours, turning occasionally. (The beef can be marinated for up to two days.) When the
beef is ready to cook, preheat the oven to 190C/375F/Gas 5. Lift the beef from the marinade, allowing any excess
liquid to drip off, and place on a plate, loosely covered, until the meat has returned to room temperature.
Sprinkle the crushed peppercorns and the remaining thyme onto a plate. Spread the mustard evenly all over the
surface of the beef, then roll the beef in the peppercorn and thyme mixture to coat. Place the crusted beef into
a roasting tin and roast in the oven for 1 hour 20 minutes (for medium-rare) or 1 hour 50 minutes (for well-
done). Meanwhile, for the horseradish cream, mix the crÃ¨me frÃ¢iche, creamed horseradish, mustard and chives
together in a bowl until well combined. Season, to taste, with salt and freshly ground black pepper, then spoon
into a serving dish and chill until needed. When the beef is cooked to your liking, transfer to a warmed platter
and cover with aluminium foil, then set aside to rest in a warm place for 25-30 minutes. To serve, carve the
rib-eye of beef into slices and arrange on warmed plates. Spoon the roasted root vegetables alongside. Serve
with the horseradish cream."
```

We also have “hand-coded” information on whether each dish is really a curry:

```r
table(recipes$curry)
```

```
    Curry Not Curry
```

Defining a curry

```
head(recipes$recipe_name[recipes$curry == "Curry"])
```

```
[1] "Venison massaman curry"             "Almond and cauliflower korma curry"
[3] "Aromatic beef curry"                "Aromatic blackeye bean curry"
[5] "Aubergine curry"                    "Bangladeshi venison curry"
```

A curry dictionary

Given that we have some idea of the concept we would like to measure, perhaps we can just use a dictionary:

```r
## Convert to corpus
recipe_corpus <- corpus(recipes, text_field = "ingredients")

# Tokenize
recipe_tokens <- tokens(recipe_corpus, remove_punct = TRUE,
                        remove_numbers = TRUE, remove_symbols = TRUE) %>%
                 tokens_remove(c(stopwords("en"),
                    "ml","fl","x","mlâ","mlfl","g","kglb",
                    "tsp","tbsp","goz","oz", "glb", "gâ", "â"))

# Convert to DFM
recipe_dfm <- recipe_tokens %>%
    dfm() %>%
    dfm_trim(max_docfreq = .3,
             min_docfreq = .002,
             docfreq_type = "prop")

topfeatures(recipe_dfm, 20)
```

```
   finely     sugar     flour    sliced    garlic    peeled       cut freerange
     3707      3118      2486      2456      2362      2333      2299      2196
   leaves     juice     white       red     large     extra    caster     seeds
     1859      1757      1730      1673      1658      1626      1615      1541
    small vegetable     onion     plain
     1498      1493      1485      1450
```

A curry dictionary

```
curry_dict <- dictionary(list(curry = c("spices",
                                        "turmeric")))

curry_dfm <- dfm_lookup(recipe_dfm, dictionary = curry_dict)

curry_dfm$recipe_name[order(curry_dfm[,1], decreasing = T)[1:10]]
```

```
 [1] "Indonesian stir-fried rice (Nasi goreng)"
 [2] "Pineapple, prawn and scallop curry"
 [3] "Almond and cauliflower korma curry"
 [4] "Aloo panchporan (Stir-fried potatoes tempered with five spices)"
 [5] "Aromatic beef curry"
 [6] "Asian-spiced rice with coriander-crusted lamb and rosemary oil"
 [7] "Beef chilli flash-fry with yoghurt rice"
 [8] "Beef rendang with mango chutney and sticky rice"
 [9] "Beef curry with jasmine rice"
[10] "Beef Madras"
```

Classification Perfomance

Let's classify a recipe as a "curry" if it includes *any* of our dictionary words

```r
recipes$curry_dictionary <- ifelse(as.numeric(curry_dfm[,1]) >

confusion_dictionary <- table(predicted_classification = recip
                              true_classification = recip
```

```r
library(caret)

confusionMatrix(confusion_dictionary, positive = "Curry")
```

```
Confusion Matrix and Statistics

                        true_classification
predicted_classification Curry Not Curry
               Curry        95       179
               Not Curry   195      8915

               Accuracy : 0.9601
                 95% CI : (0.956, 0.964)
    No Information Rate : 0.9691
    P-Value [Acc > NIR] : 1.000

                  Kappa : 0.3164

 Mcnemar's Test P-Value : 0.438

            Sensitivity : 0.32759
            Specificity : 0.98032
         Pos Pred Value : 0.34672
         Neg Pred Value : 0.97859
```

$$\text{Accuracy} = \frac{\#\text{True Positives} + \#\text{T}}{\#\text{Observat}}$$

$$\text{Sensitivity} = \frac{\#\text{True Po}}{\#\text{True Positives} + \#}$$

$$\text{Specificity} = \frac{\#\text{True Neg}}{\#\text{True Negative} + \#}$$

Implication:

- We can pick up some signal with the dictionary, but we are not doing a great job of classifying curries
- Our sensitivity is a very low
- We need methods that are better at working out the

Supervised Learning vs Dictionaries

Supervised learning methods classify documents into pre-defined categories on the basis of the words they contain.

- Supervised learning can be conceptualized as a generalization of dictionary methods
- Dictionaries:
  - Words associated with each category are pre-specified by the researcher
  - Words typically have a weight of either zero or one
  - Documents are scored on the basis of words they contain
- Supervised learning:
  - Words are associated with categories on the basis of pre-labelled training data
  - Words have are weighted according to their relative prevalence in each each category
  - Documents are scored on the basis of words they contain
- The key difference is that in supervised learning the features associated with each category (and their relative weight) are **learned** from the data

Components of Supervised Learning

- Labelled dataset
  - Labelled (normally hand-coded) data which categorizes texts into different categories
  - Training set: used to train the classifier
  - Test set: used to validate the classifier
- Classification method
  - Statistical method to:
    - learn the relationship between coded texts and words
    - predict unlabeled documents from the words they contain
  - Examples: Naive Bayes, Logistic Regression, SVM, tree-based methods, many others…
- Validation method
  - Predictive metrics such as confusion matrix, accuracy, sensitivity, specificity, etc
  - Normally we use a specific type of validation known as *cross-validation*

Creating a labelled datset

How do we obtain a labelled set?

- External sources of annotation, e.g.
    - Party labels for election manifestos
    - Disputed authorship of Federalist papers estimated based on known authors of other documents
- Expert annotation, e.g.
    - In many projects, undergraduate students ("expertise" comes from training)
    - Existing expert annotations, e.g. Comparative Manifesto Project
- Crowd-sourced coding, e.g.
    - Ask random people on the internet to code texts into categories
    - Tends to rely on the "wisdom of crowds" hypothesis: aggregated judgments of non-experts converge to judgments of experts at much lower cost

For the purposes of the running example, we are cheating a bit by assuming that any dish whose title contains the word "curry" is, in fact, a curry.

Naive Bayes Classification

Language Models

- Probabilistic language models describe a story about how documents are generated using probability
- This *data-generating process* is based on a set of unknown parameters which we infer based on the data
- Once we have inferred values for the parameters, we can reverse the data-generating process and calculate the probability that any given document was generated by a particular language model
- The Naive Bayes text classification model is *one* example of a generative language model. In Naive Bayes:
  a. Estimate separate language models for each category of interest
  b. Calculate probability that each text was generated by each model
  c. Assign the text to the category for which it has the highest probability

Language Models

- The basis of any language model is a probability distribution over words in a vocabulary.
- A probability distribution over a discrete variable must have three properties
  - Each element must be greater than or equal to zero
  - Each element must be less than or equal to one
  - The sum of the elements must be 1

Language Models

- Consider a 6 word vocabulary: “coriander”, “turmeric”, “garlic”, “sugar”, “flour”, “eggs”
- When writing a curry recipe, you will
  - frequently use the words “coriander”, “turmeric”, and “garlic”
  - infrequently use the words “sugar”, “flour”, and “eggs”
- When writing a cake recipe, you will
  - frequently use the words “sugar”, “flour”, and “eggs”
  - infrequently use the words “coriander”, “turmeric”, and “garlic”
- We can represent these different “models” for language using a probability distribution over the words in the vocabulary:

Model	**coriander**	**turmeric**	**garlic**	**sugar**	**flour**	**eggs**
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

- Given these models, we can calculate the probability that a given set of word counts (i.e. a document) would be drawn from each distribution

$$P(W_i|\mu) = \frac{M_i!}{\prod_{j=1}^{J} W_{i,j}!} \prod_{j=1}^{J} \mu_j^{W_{ij}}$$

- This is the **multinomial** distribution
- μ_j is the probability of observing word j under a given model
- $W_{i,j}$ is the number of times word j appears in document i (i.e. it is an element of a dfm)
- M_i is the total number of words in document i

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Which language model is most likely to have produced each document?

Naive Bayes

- Naive Bayes is a model that classifies documents into categories on the basis of the words they contain

- is the **posterior distribution** – this tells us the probability that document is in category , given the words in the document and the prior probability of category

- is the **conditional probability** or **likelihood** – this tells us the probability that we would observe the words in if the document *were* from category

- is the **prior probability** that the document is from category – this tells us the probability of the category of the document, absent any information about the words it contains

- is the **unconditional probability** of the words in document – this tells us the probability that we would observe the words in across all categories

Naive Bayes

- Generally, we will want to make comparisons of the probabilities between different classes
  - e.g. Is
- This means that we can drop the term and just focus on the likelihood and the prior probabilities
- where means “proportional to” (rather than “equal than” for)

Naive Bayes

To work out the whether a document should be labelled as belonging to a particular class, we therefore need to work out:

- the **prior probability** () that the document is from category
  - This is usually estimated by calculating the proportion of documents of category in the training data
- the **conditional probability** or **likelihood** () of the words in the document occuring in category
  - We already know that we can calculate this probability from the multinomial distribution!
  - Again, because we are only interested in the relative probabilities of different classes, we can drop the multinomial coefficient

Question: How do we estimate ?

Naive Bayes Estimation

- is the probability that word will occur in documents of category .
- We can **estimate** these probabilities from our training data:

Example:

- In the **curry recipes** our training data, we observe…
  - …77 instances of the word “turmeric” ()
  - …10586 total words ()
  - …and so
- In the **not-curry recipes** our training data, we observe…
  - …148 instances of the word “turmeric” ()
  - …210805 total words ()
  - …and so
- The word “turmeric” is about 10 times more common in curry recipes than other recipes

Naive Bayes Estimation – Laplace Smoothing

- What happens when a given word doesn’t appear at all for one of the classes in our training data?
- Imagine that we never observe the word “duck” in the curry recipes in our training data
- Then, in our test data, we observe the following sentence:

```
> "For this curry you will need to coat the duck legs with 1 tsp ground turmeric"
```

- Because we multiply together all the individual word probabilities when we calculate the probability of a sentence occurring in a category, we will get a probability of zero!
- **Solution**: Add one to the counts for each word in each category
- This solution is known as “add-one” or “Laplace” smoothing

Why is Naive Bayes “Naive”?

By treating documents as bags of words we are assuming:

- *Conditional independence* of word counts
    - Knowing a document contains one word doesn’t tell us anything about the probability of observing other words in that document
    - e.g. The fact that a recipe includes the word “turmeric” doesn’t make it any more or less likely that it will also include the word “coriander”
- *Positional independence* of word counts
    - The position of a word within a document doesn’t give us any information about the category of that document
    - e.g. Whether the word “turmeric” appears early or late in the recipe has no effect on the probability of it being a curry

While this is a very simple model of language which is “wrong”, it is nevertheless useful for classification.

Naive Bayes Classification

The classification decision made by the Naive Bayes model is simple: we assign document to the category, , for which it has the highest posterior probability:

where means “which category, , has the *maximum* posterior probability”.

Intuition:

- Assign documents to categories when the probability of observing the words in that document are high given the probability distribution for that category (i.e. when is large)
- Assign more documents to categories that contain more documents in the training data (i.e. when is large)

Naive Bayes Application

```r
nb_output <- textmodel_nb(x = recipe_dfm,
                          y = recipe_dfm$curry,
                          prior = "docfreq")
summary(nb_output)
```

```

Call:
textmodel_nb.dfm(x = recipe_dfm, y = recipe_dfm$curry, prior = "docfreq")

Class Priors:
(showing first 2 elements)
    Curry Not Curry
   0.0309    0.9691

Estimated Feature Scores:
              beef    boned   rolled     pint      red     wine  vinegar
Curry     0.001378 0.0014925 0.0001148 0.001148 0.011481 0.001607 0.001378
Not Curry 0.003304 0.0006107 0.0004031 0.003847 0.009619 0.007750 0.005154
            sugar  allspice      bay  leaves     thyme peppercorns  crushed
Curry     0.00620 0.0002296 0.002067 0.01378 0.0004592    0.003789 0.009070
Not Curry 0.01872 0.0003420 0.002821 0.01063 0.0048734    0.001826 0.005417
            english     dijon  mustard unsalted      room temperature      lard
Curry     0.0002296 0.0001148 0.005166 0.001493 0.0001148   0.0001148 0.0001148
Not Curry 0.0007023 0.0009832 0.002803 0.004953 0.0005741   0.0005924 0.0004153
             plain    flour   white    water   chilled     icing  chicken
Curry     0.004822 0.006085 0.00287 0.007003 0.0001148 0.0003444 0.005855
Not Curry 0.008611 0.014871 0.01042 0.006693 0.0005863 0.0023573 0.007035
              cut   pieces
Curry     0.01297 0.005511
Not Curry 0.01336 0.003786
```

Naive Bayes Application

Recall that we are interested in the probability of observing word given class , i.e.

What are these word probabilities for our curry data?

We can examine the probability of each word given each class using the `coef()` function on the `nb_train` object.

```
head(coef(nb_output))
```

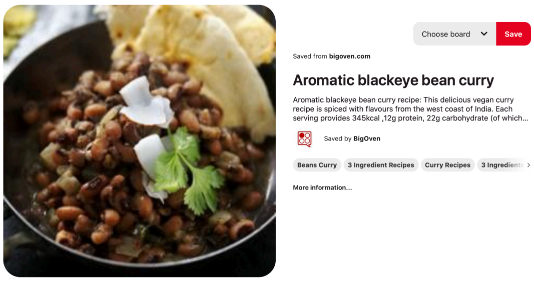
```
              Curry    Not Curry
beef   0.0013777268 0.0033038975
boned  0.0014925373 0.0006107019
rolled 0.0001148106 0.0004030633
pint   0.0011481056 0.0038474222
red    0.0114810563 0.0096185556
wine   0.0016073479 0.0077498076
```

Naive Bayes Application

Naive Bayes Application

What are the class-conditional word probabilities for “Aromatic blackeye bean curry”?

	P(w|curry)	P(w|not curry)
seeds	0.030	0.008
finely	0.023	0.021
coriander	0.021	0.005
peeled	0.018	0.013
garlic	0.017	0.014
ginger	0.015	0.005
cloves	0.015	0.008
leaves	0.014	0.011
cumin	0.014	0.002
chilli	0.013	0.006
onion	0.010	0.009
piece	0.010	0.002



What are the class-conditional word probabilities for “Schichttorte”?

	P(w|curry)	P(w|not curry)
large	0.010	0.010
sugar	0.006	0.019
flour	0.006	0.015
paste	0.006	0.001
plain	0.005	0.009
lemon	0.004	0.008
freerange	0.003	0.013
eggs	0.002	0.008

Naive Bayes Application

Which recipes are predicted to have a high curry probability?

```r
recipe_dfm$curry_nb_probability <- predict(nb_output,
                                           type = "probability")

recipe_dfm$recipe_name[order(recipe_dfm$curry_nb_probability[,1], decreasing = T)[1:10]]
```

```
 [1] "Bengali butternut squash with chickpeas"
 [2] "Chickpea curry with green mango and pomegranate"
 [3] "Green coconut fish curry"
 [4] "Thai green prawn curry"
 [5] "Rogan josh"
 [6] "Bengal coconut dal"
 [7] "Tom yum soup"
 [8] "Thai-style duck red curry"
 [9] "Peppery hot cabbage salad"
[10] "Peppery hot cabbage salad"
```

Which recipes are predicted to have a low curry probability?

```r
recipe_dfm$recipe_name[order(recipe_dfm$curry_nb_probability[,1], decreasing = F)[1:10]]
```

```
 [1] "Sticky toffee apple pudding with calvados caramel sauce"
 [2] "Rich moist all-purpose fruit cake"
 [3] "Mini stollen "
 [4] "Chocolate fruit cake"
 [5] "Pheasant pithiviers"
 [6] "Spiced poached pears with chocolate pudding"
 [7] "Traditional Christmas pudding with brandy butter"
 [8] "Intense chocolate cookies"
 [9] "Cookies and cream fudge brownies"
[10] "Bonfire night brioche"
```

Was #TheStew really #TheCurry?

- The purpose of training a classification model is to make **out-of-sample** predictions
- Generally, we have a small hand-coded training dataset and then we predict for lots of other documents
- Here, we are only predicting for one out-of-sample observation

```
ingredients <- c("cup olive oil, plus more for serving garlic cloves, chopped large yellow onion, chopped (2-

dfm_stew <- tokens(ingredients) %>%
            dfm() %>%
            dfm_match(features = featnames(recipe_dfm))

predict(nb_output, newdata = dfm_stew, type = "probability")
```

```
          Curry  Not Curry
text1 0.9611718 0.03882815
```

Yes!

Advantages and Disadvantages of Naive Bayes

Advantages

- Fast
  - Takes seconds to compute, even for very large vocabularies/corpuses
- Easy to apply
  - One line of code in quanteda
- Can easily be extended to include…
  - … multiple categories
  - … different text representations (bigrams, tri-grams etc)

Advantages and Disadvantages of Naive Bayes

Disadvantages

- Independence assumption
    - Independence means NB is unable to account for interactions between words
        - e.g. When the word “eggs” appears with the word “sugar” that should indicate something different from when “eggs” appears without the word “sugar”
    - Independence also means that NB is often overconfident
        - Each additional word counts as a new piece of information
    - In some contexts, the independence assumption can decrease predictive accuracy
- Linear classifier
    - Other methods (e.g. SVM) allow the classification probabilities to change *non-linearly* in the word counts
    - e.g. Perhaps seeing the word “eggs” once should have a smaller effect on the probability that the recipe is a curry than seeing the word “eggs” five times

Validation

Naive Bayes Application

Before we train a model, we need to separate our data into a training set and a test set:

```
## Training and test set

train <- sample(c(TRUE, FALSE), nrow(recipes), replace = TRUE, prob = c(.8, .2))
test <- !train
```

```
table(train)
```

```
train
FALSE  TRUE
 1877  7507
```

```
table(test)
```

```
test
FALSE  TRUE
 7507  1877
```

How many curry recipes are there in the training and test sets?

```
## Training and test set

prop.table(table(recipes$curry[train]))
```

```
     Curry  Not Curry
0.03157053 0.96842947
```

```
prop.table(table(recipes$curry[test]))
```

```
     Curry  Not Curry
0.02823655 0.97176345
```

Naive Bayes Application

We then subset the `recipe_dfm` object into a training dfm and a test dfm:

```
## Naive Bayes

recipe_dfm_train <- dfm_subset(recipe_dfm, train)
recipe_dfm_test <- dfm_subset(recipe_dfm, test)
```

We then train our Naive Bayes model on the training set:

```
nb_train <- textmodel_nb(x = recipe_dfm_train,
                         y = recipe_dfm_train$curry,
                         prior = "docfreq")
```

And finally, we predict the category of each recipe in the test set:

```
recipe_dfm_test$predicted_curry_nb <- predict(nb_train,
                                              newdata = recipe_dfm_test,
                                              type = "class")
```

Naive Bayes Classification Perfomance

```r
confusion_nb <- table(predicted_classification = recipe_dfm_te
                      true_classification = recipe_dfm_test$cu
```

```r
library(caret)

confusionMatrix(confusion_nb, positive = "Curry")
```

```
Confusion Matrix and Statistics

                        true_classification
predicted_classification Curry Not Curry
               Curry        38       101
               Not Curry    15      1723

               Accuracy : 0.9382
                 95% CI : (0.9263, 0.9487)
    No Information Rate : 0.9718
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3701

 Mcnemar's Test P-Value : 2.973e-15

            Sensitivity : 0.71698
            Specificity : 0.94463
         Pos Pred Value : 0.27338
         Neg Pred Value : 0.99137
             Prevalence : 0.02824
         Detection Rate : 0.02025
   Detection Prevalence : 0.07405
```

Implication:

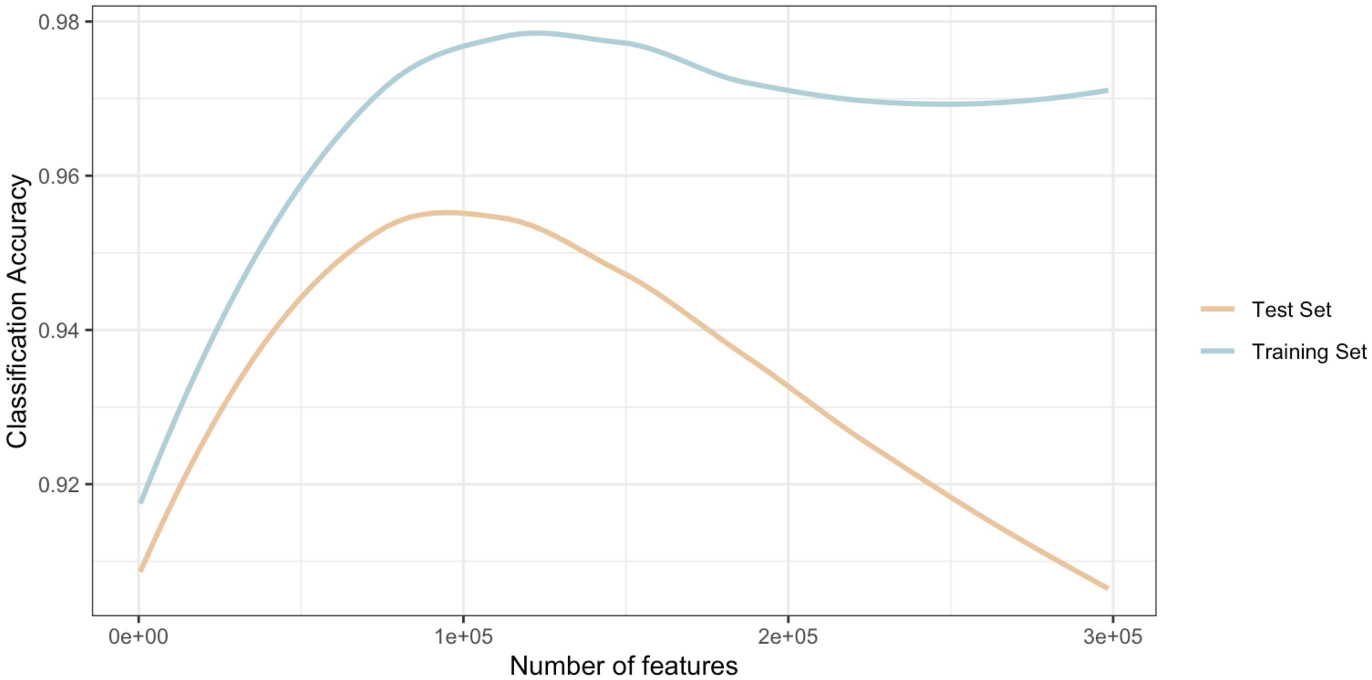
Relative to the dictionary approach we are…

- …doing a better job on predicting true positives now (our sensitivity is much higher)
- …predicting too many curries that are actually something else (our specificity is a little lower)

Training-Set and Test-Set Performance

- The test set and training set accuracy can be very different
- As a model becomes more flexible…
  - …the training set accuracy will almost always increase
  - …the test set accuracy will sometimes decrease
- Imagine that we include a very large number of features in our dfm
  - All unigrams, all bi-grams, …, all 5-grams
  - Total number of features 300k features
- How does the training/test set accuracy change as we increase the number of features used to train the classifier?

Training-Set and Test-Set Accuracy



Overfitting and Test-Set Accuracy

- **Question:** Why does the test-set accuracy decrease when we add additional features?
- **Answer:** Because we are now *overfitting* our data.
- Overfitting occurs when we find relationships between words (or n-grams) and curries in our training data that do not generalise to our test data
- In this example, there are some n-gram phrases that appear frequently in the curry recipes in our training set but which never appear in our test-set curry recipes

Feature	Trai
mustard_seeds_tsp	
tsp_black_mustard	
tsp_black_mustard_seeds	
leaves_and_stalks	
black_mustard_seeds_tsp	
coriander_leaves_and	
cumin_seeds_tsp_black	
coriander_leaves_and_stalks	
large_garlic_cloves	
chopped_garlic_cloves_peeled_and	

Test-Set Validation for Feature Selection

- We can use the test-set performance statistics to select between model specifications
- We will compare the accuracy, sensitivity and specificity for the following models:
  - Our “original” model (unigrams, no stopwords, trimmed)
  - A “raw” model (unigrams, nothing removed)
  - A “no stopwords” model (unigrams, stopwords removed)
  - A “trimmed” model (unigrams, trimmed)
  - An “n-gram” model (unigrams, bigrams, trigrams)
  - An “n-gram, trimmed” model (unigrams, bigrams, trigrams, words occuring fewer than 10 times discarded)
- The “best” model is the one which has the highest classification scores

Test-Set Validation for Feature Selection

Test-set validation

Model	Accuracy	Sensitivity	Specificity	N features
Original	0.94	0.78	0.95	902
Raw	0.96	0.65	0.97	4214
No stop words	0.96	0.66	0.97	4126
Trimmed	0.94	0.79	0.95	1339
N-gram	0.98	0.51	1	152215
N-gram, trimmed	0.94	0.83	0.94	6072

- The “n-gram” model has the highest accuracy, but has very low sensitivity
- The “n-gram, trimmed” model outperforms all other models in sensitivity

Cross-Validation

- To calculate the test-set accuracy we **randomly** allocated observations to the test and training sets
- If we repeat this process with a new randomization, we will get slightly different test-set performance scores

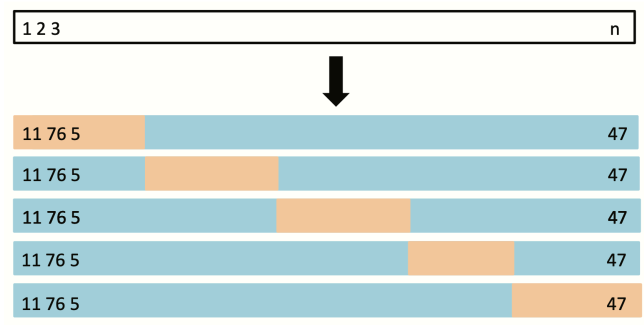
Rerandomization 3:

Test-set validation

Model	Accuracy	Sensitivity	Specificity	N features
Original	0.94	0.78	0.95	902
Raw	0.96	0.64	0.97	4214
No stop words	0.96	0.69	0.97	4126
Trimmed	0.94	0.78	0.95	1339
N-gram	0.98	0.51	1	152215
N-gram, trimmed	0.94	0.83	0.94	6072

K-fold Cross-Validation

- Cross-validation is an alternative to a simple train-test split
- This approach involves randomly dividing the set of observations into groups, or *folds*, of approximately equal size
  - Typical choices are or
- For each of the folds we do the following
  1. Train the Naive Bayes model on all observations not included in the fold
  2. Generate predictions for the observations in the fold
  3. Calculate the accuracy etc of the predictions for the observations in the held-out fold



K-fold Cross-Validation Application

```r
get_performance_scores <- function(held_out){

  # Set up train and test sets for this fold
  recipe_dfm_train <- dfm_subset(recipe_dfm, !held_out)
  recipe_dfm_test <- dfm_subset(recipe_dfm, held_out)

  # Train model on everything except held-out fold
  nb_train <- textmodel_nb(x = recipe_dfm_train,
                           y = recipe_dfm_train$curry,
                           prior = "docfreq")

  # Predict for held-out fold
  recipe_dfm_test$predicted_curry <- predict(nb_train,
                                             newdata = recipe_dfm_test,
                                             type = "class")

  # Calculate accuracy, specificity, sensitivity
  confusion_nb <- table(predicted_classification = recipe_dfm_test$predicted_curry,
                        true_classification = recipe_dfm_test$curry)

  confusion_nb_statistics <- confusionMatrix(confusion_nb, positive = "Curry")

  accuracy <- confusion_nb_statistics$overall[1]
  sensitivity <- confusion_nb_statistics$byClass[1]
  specificity <- confusion_nb_statistics$byClass[2]

  return(data.frame(accuracy, sensitivity, specificity))
```

K-fold Cross-Validation Application

```
K <- 5
folds <- sample(1:K, nrow(recipe_dfm), replace = T)
get_performance_scores(folds == 1)
```

```
         accuracy sensitivity specificity
Accuracy 0.9418182    0.754386   0.9475375
```

```
all_folds <- lapply(1:5, function(k) get_performance_scores(folds == k))
all_folds
```

```
[[1]]
         accuracy sensitivity specificity
Accuracy 0.9418182    0.754386   0.9475375

[[2]]
         accuracy sensitivity specificity
Accuracy 0.9389356   0.6923077   0.9463358

[[3]]
        accuracy sensitivity specificity
Accuracy 0.935911   0.7666667   0.9414661

[[4]]
         accuracy sensitivity specificity
Accuracy 0.9420829   0.7704918   0.9478309

[[5]]
         accuracy sensitivity specificity
Accuracy 0.9380252   0.7166667   0.9452278
```

```
colMeans(bind_rows(all_folds))
```

```
   accuracy sensitivity specificity
  0.9393546   0.7401038   0.9456796
```

Cross-Validation for Model Selection

Extensions

Naive Bayes is only one supervised learning text-classification method

- Regularized Logistic Regression
  - Directly models the probability that each document is in class using logistic regression
  - Regularization required to prevent overfitting data
  - `textmodel_lr` in `quanteda`
- Support Vector Machines
  - SVMs draw a hyperplane through the multidimensional word space that best separates documents into different classes
  - Can accomodate *non-linear* boundaries between classes
  - `textmodel_svm()` in `quanteda`
- "Tree-based" Classification Methods
  - Tree-based methods separate classes by segmenting the predictors (word counts) into a number of distinct regions

Use Cases of Supervised Learning

Conclusion

Summing Up

1. Using a vector-based representation allows us to calculate the **similarity** between documents
2. Supervised learning for text data allows us to learn the association between words and particular outcome categories
3. The Naive Bayes model is a simple model that is fast to implement and which, despite some strong assumptions, tends to provide good classification results