UK Data Service

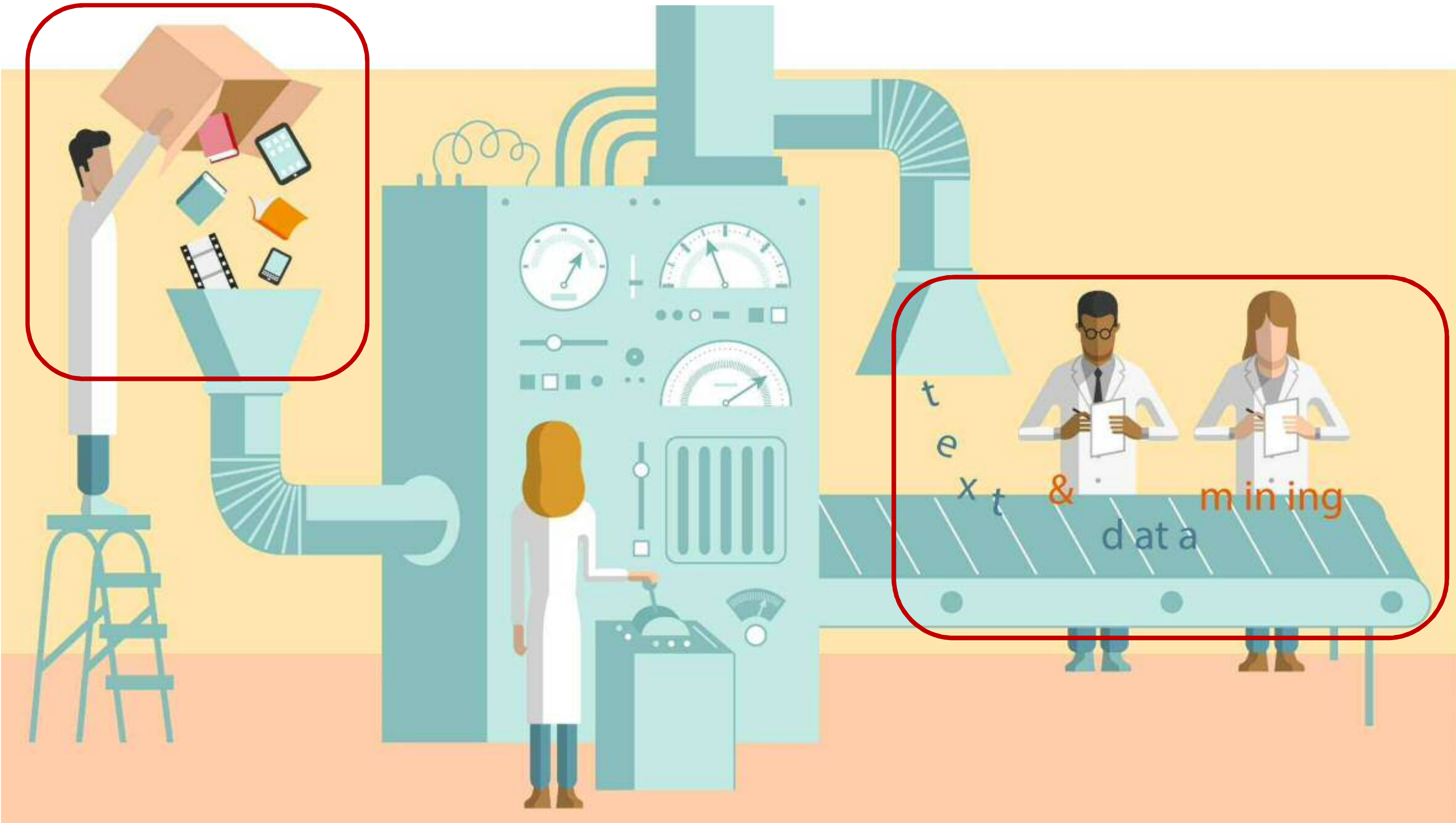
Text-Mining: Basic Processes

Dr. J. Kasmire
Research Fellow at Cathie Marsh Institute and UK Data Service

julia.kasmire@manchester.ac.uk

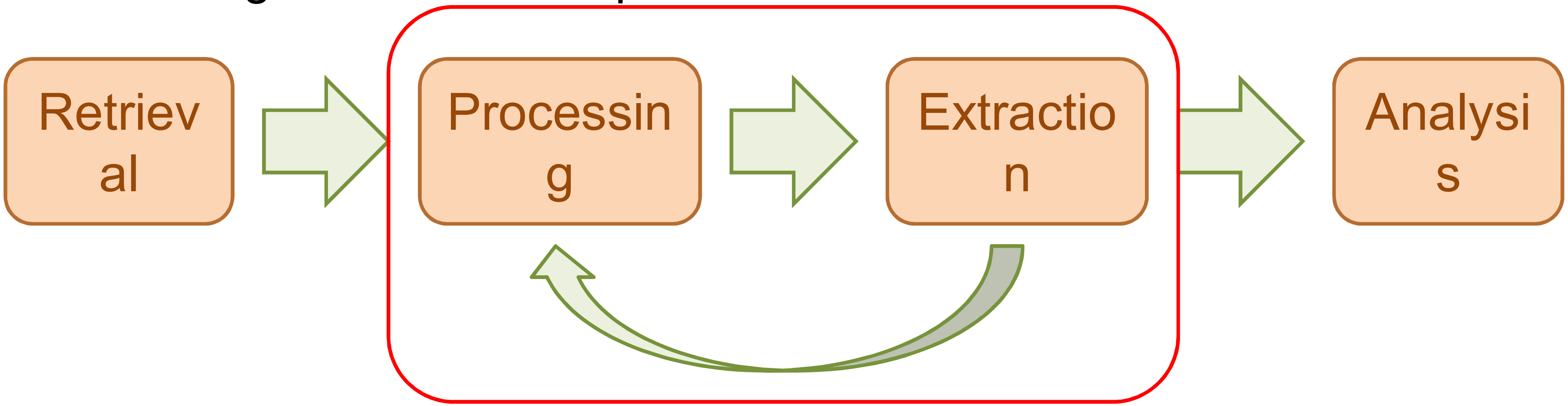
@JKasmireComplex

Text-mining is a form of data-mining



UK Data Service

Text-mining has 4 basic steps



Processing:

- Tokenisation (dividing raw data)
- Standardising (case, spelling, RegEx)
- Removing irrelevancies (punctuation, stopwords, etc.)
- Consolidation (stemming and/or lemmatising)

Basic NLP:

- Tagging, Named Entity Recognition and Chunking

Basic Extraction:

- POS-tagging
- Chunking
- Named Entity Recognition
- Word frequency
- Similarity
- Discovery

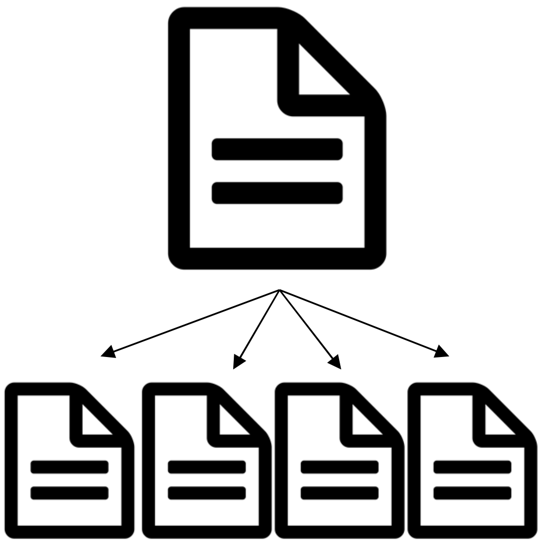
UK Data Service

Processing – Raw data into useful data

Great big file with the text content of hundreds of newspaper articles.

You may want to:

- Break it into many small files of one article each (with useful names)
- Insert a line break after each article
- Write out each article to a dictionary with key-value pairs for article features



Semantic Line Breaks

[‘Author(s)’: ‘Writer1, Writer2’
‘Date’: ‘Junetember 43, 3024’
‘Headline’: ‘They started Text-Mining and you will not believe what happens next!’
‘Publication’: ‘Fake News Corp.’
‘Article’: ‘Yada yada yada, blah.’]

UK Data Service

Processing – Tokenisation

Tokens = lowest unit of natural language processing analysis.

Example:
text = "It's raining cats and dogs. It is also raining elephants, which is becoming a problem."

Tokenize by words

['It' "'s" 'raining' 'cats' 'and' 'dogs' '.' 'It' 'is' 'also' 'raining' 'elephants' ',' 'which' 'is' 'becoming' 'a' 'problem' '.']

Tokenize by sentences

["It's raining cats and dogs." 'It is also raining elephants, which is becoming a problem.']

UK Data Service

Processing – Standardising

Goal is to replace multiple forms of ‘same’ token with a single form

RegEx is like find-and-replace - useful for standardising on terminology/acronyms/etc.

Example: “cats" --> “puddy-tats"

‘It's raining cats and dogs. It is also raining elephants, which is becoming a problem.’

‘It's raining puddy-tats and dogs. It is also raining elephants, which is becoming a problem.’

Processing – Standardising

Multiple replacements with a RegEx dict = {‘cats’ : ‘puddy-tats’,
‘dogs’ : ‘doggos’,
‘elephants’ : ‘rhinos’,
‘problem’ : ‘kerfuffle’, }

‘It's raining cats and dogs. It is also raining elephants, which is becoming a problem.’

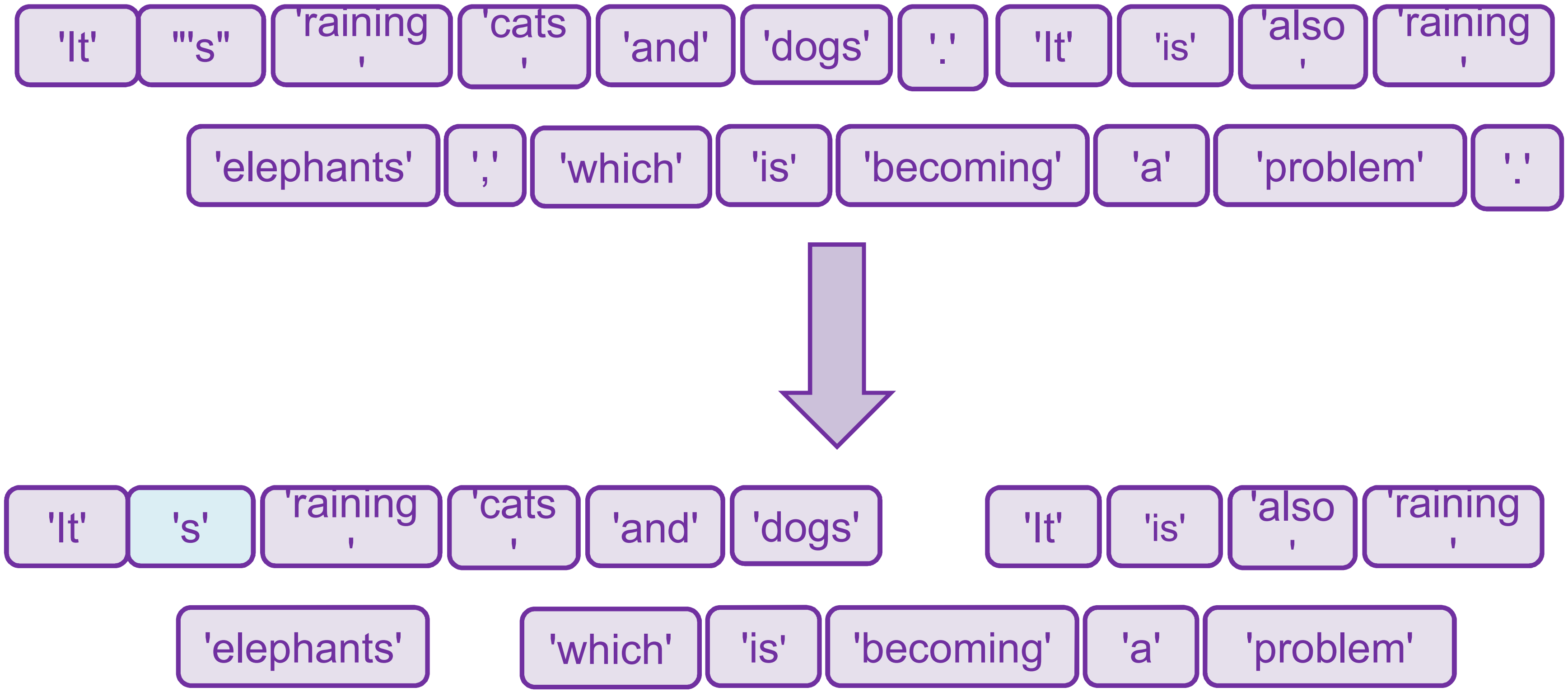
‘It's raining puddy-tats and doggos. It is also raining rhinos, which is becoming a kerfuffle.’

Many standardisation tools with different targets

UK Data Service

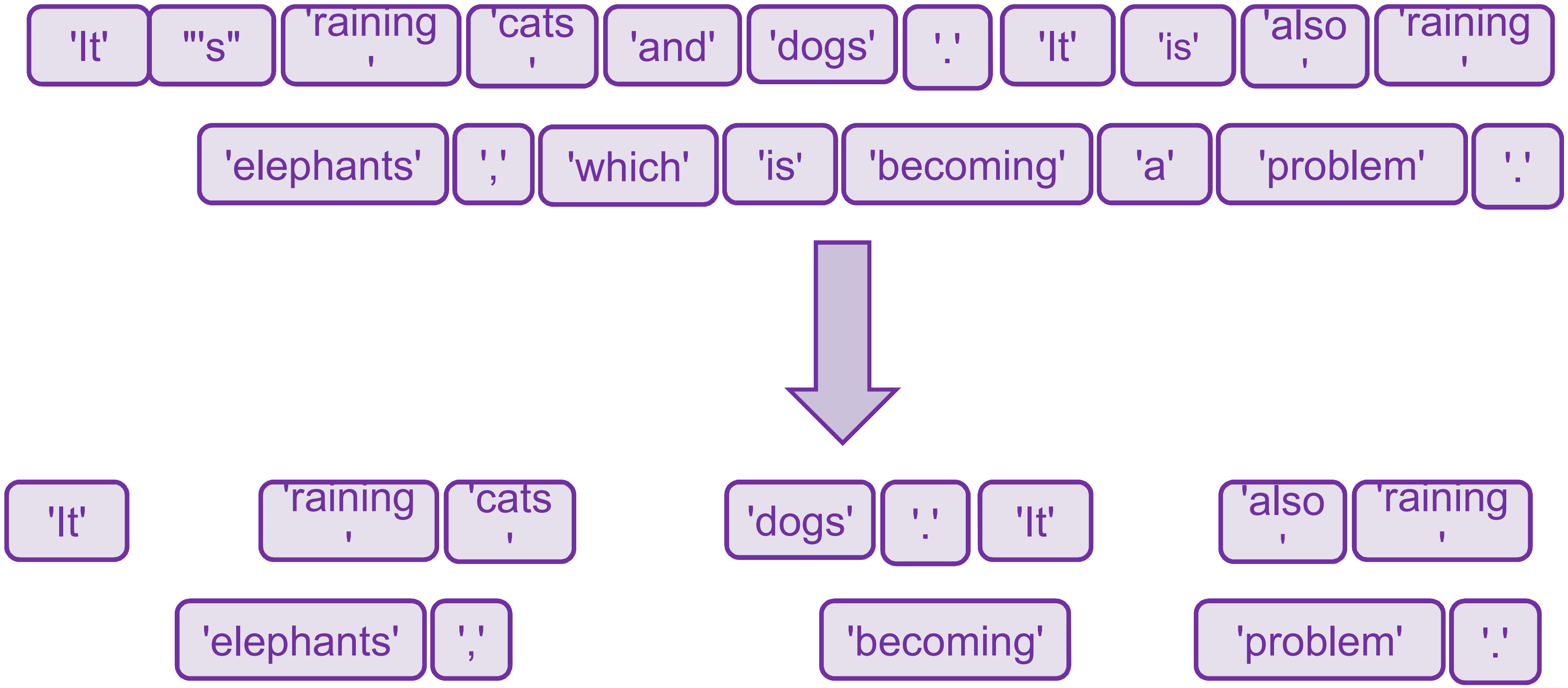
Processing – Removing irrelevancies

Punctuation



Processing – Removing irrelevancies

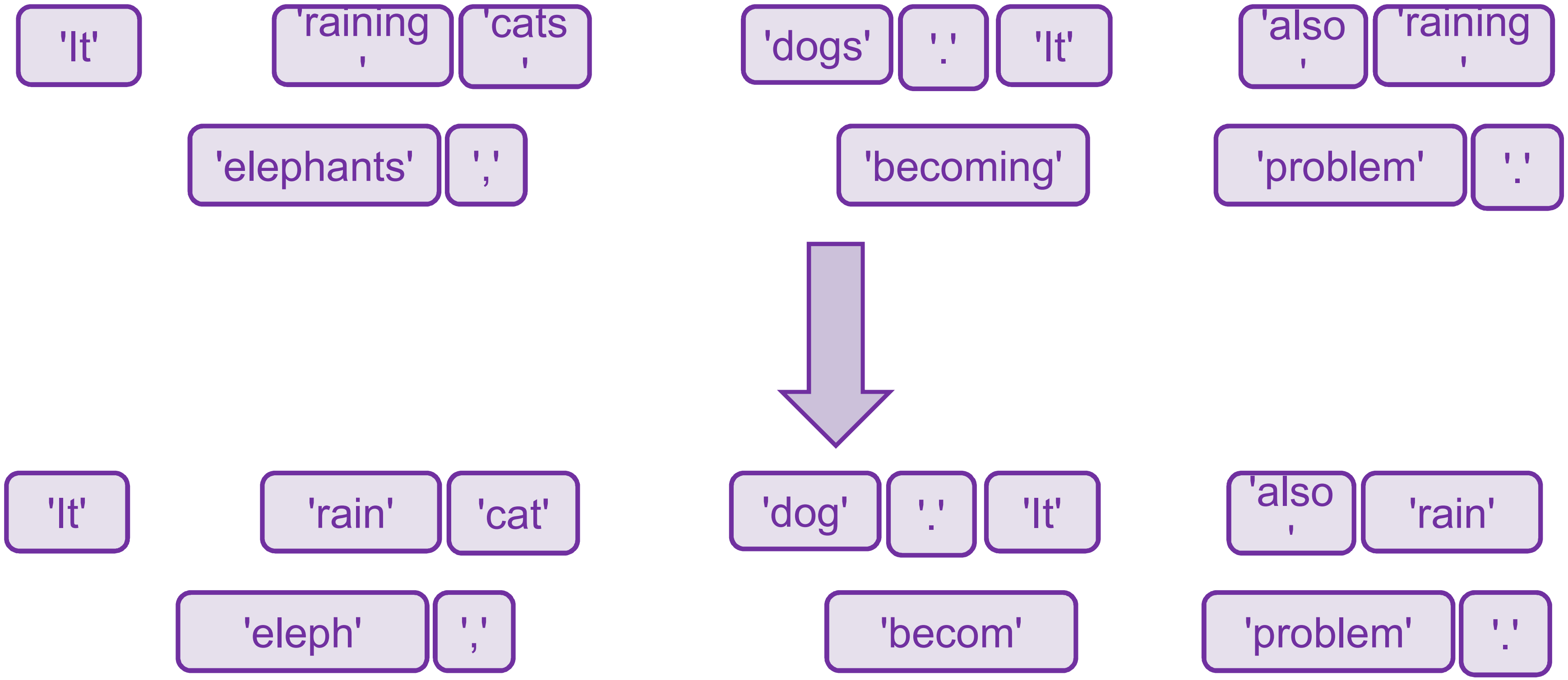
Stop words



UK Data Service

Processing – Consolidation

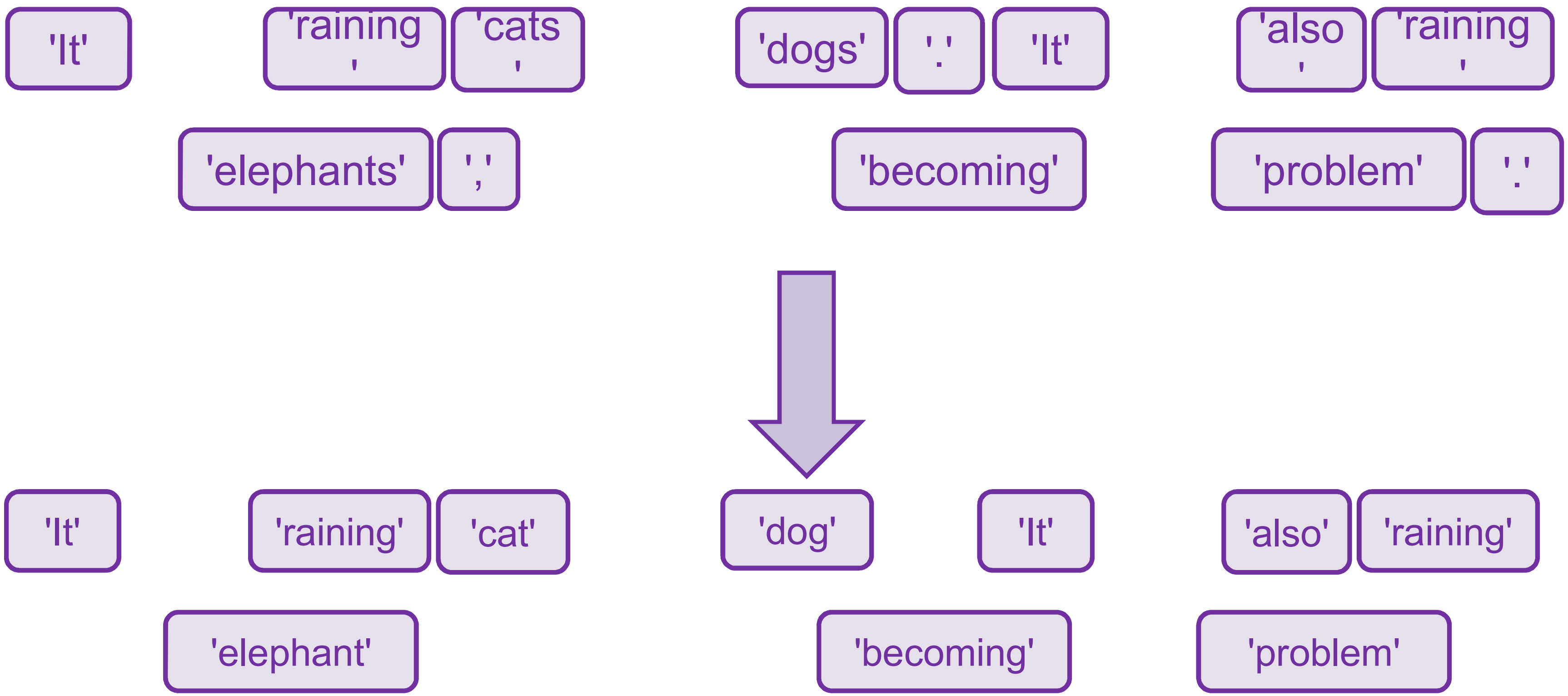
Removing different word forms so they count as ‘the same word’
Stemming



UK Data Service

Processing – Consolidation

Lemmatising



UK Data Service

Basic NLP – Part of Speech tagging

'It' 'raining' 'cats' 'dogs' 'It' 'also' 'raining'

'elephants' 'becoming' 'problem'

('It', 'PRN') ('raining', 'VBG') ('cats', 'NNS') ('dogs', 'NNS') ('It', 'PRN') ('also', 'RB')

('raining', 'VBG') ('elephants', 'NNS') (' becoming', 'VBG') ('problem', 'NN')

Basic NLP – Post POS-tagging Lemmatisation

('It', 'PRN') | ('raining', ‘VBG’) | (‘cats', ‘NNS’) | (‘dogs', ‘NNS’) | ('It', 'PRN’) | (‘also', ‘RB’)

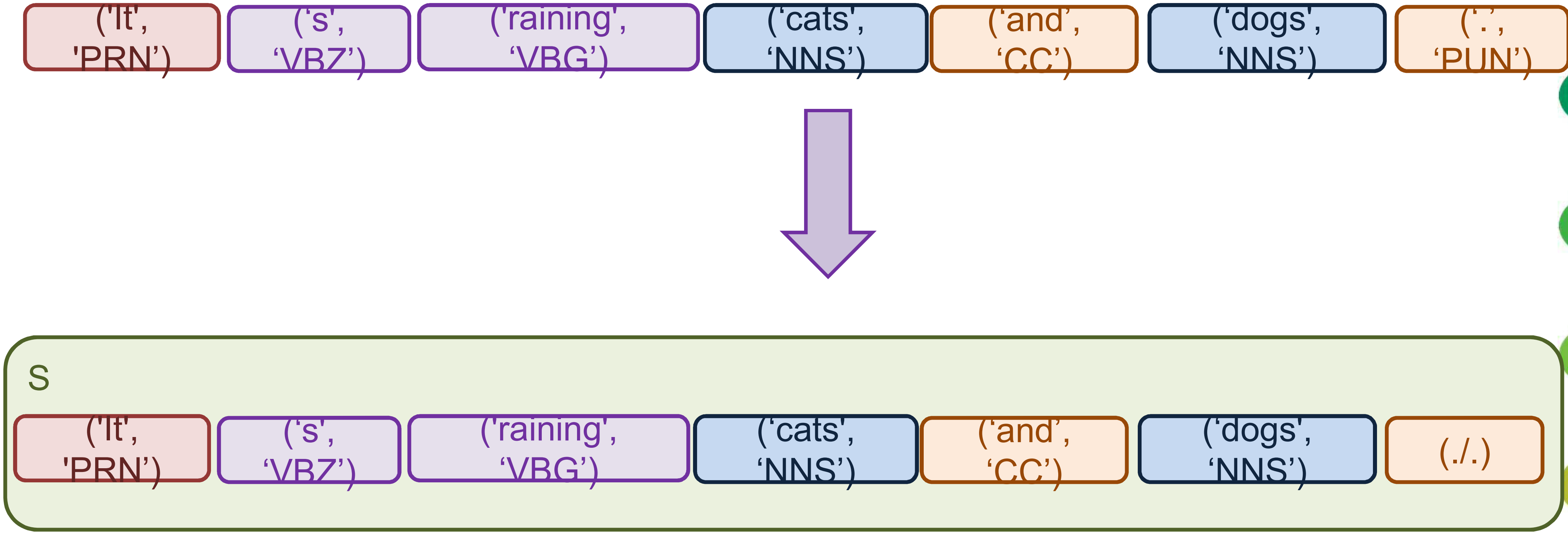
('raining', ‘VBG’) | (‘elephants', ‘NNS’) | (' becoming', ‘VBG’) | (‘problem', ‘NN’)

↓

'It' | 'rain' | ‘cat' | ‘dog' | 'It' | ‘also'

'rain' | ‘elephant' | 'become' | ‘problem'

Basic NLP – Chunking



(S It/PRP 's/VBZ raining/VBG cats/NNS and/CC dogs/NNS ./.)

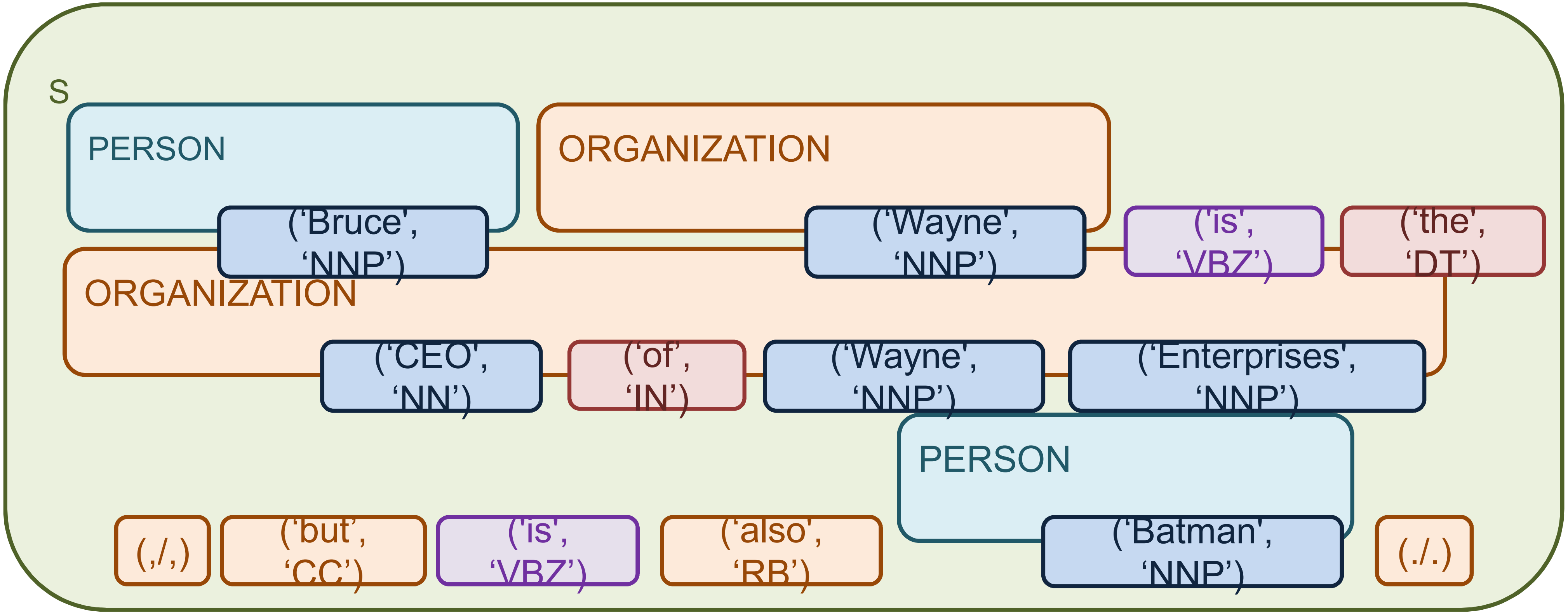
Basic NLP – Named Entity Recognition

(‘Bruce’, ‘NNP’) (‘Wayne’, ‘NNP’) (‘is’, ‘VBZ’) (‘the’, ‘DT’) (‘CEO’, ‘NN’) (‘of’, ‘IN’)

(‘Wayne’, ‘NNP’) (‘Enterprises’, ‘NNP’) (‘,’, ‘PUN’) (‘but’, ‘CC’) (‘is’, ‘VBZ’) (‘also’, ‘RB’)

(‘Batman’, ‘NNP’) (‘.’, ‘PUN’)

Basic NLP – Named Entity Recognition



Processing – What to do and in what order?

Chunking and/or POS-lemmatising requires text that is already tokenised and POS-tagged.

RegEx may be best before removing uppercase to better catch acronyms or abbreviatons.

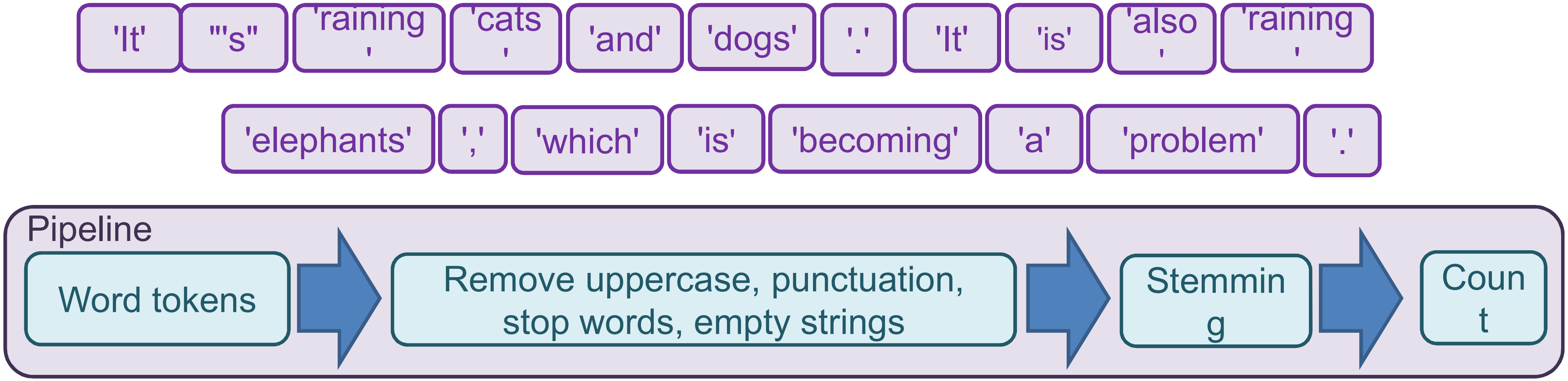
Add changes to a pipeline and run the whole thing from scratch.

Replicability is important!

Pipeline

Tokenisation → POS-tagging → Lemmatising → Remove punctuation/stop words

Extraction - Word Frequency



Example:
{'It': 2, 'raining': 2,

'cats': 1, 'dogs': 1, 'also': 1, 'elephants': 1, 'becoming': 1, 'problem': 1}

Extraction - Word Frequency

The entire text of ‘Emma’ by Jane Austen
(available through nltk.corpus.gutenberg functions)

Pipeline

Word tokens → Remove uppercase, punctuation, stop words, empty strings → Stemmin g → Coun t

10 most common words =
{'mr', 1855, 'emma', 865, 'could', 837, 'would', 821, 'miss', 614, 'must', 571, 'harriet', 506, 'much', 486, 'said', 484, 'think', 467}

Count of the word ‘common’ = 142

UK Data Service

Extraction – Word similarity

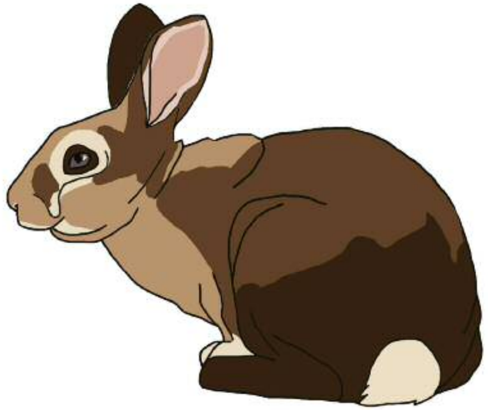
Uses concepts of 'word vectors' (built into packages like spaCy)

Score included words on 300 dimensions derived from

- How the word is used in large corpora of natural language
- Part of speech, etc.
- What words are typically found before or after
- Etc.

Word-to-Word similarity returns a score between 0 (no similarity) and 1 (identical).

Extraction – Word similarity



	TROLL	ELF	RABBIT
TROLL	1	0.4	0.29
ELF	0.4	1	0.34
RABBI	0.29	0.34	1

Extraction – Document similarity

Document similarity works in a comparable way:

- Document vectors are created (no pre-loaded document vectors)
- 2 or more document vectors are compared
- Returns value between 0 and 1

- ‘Emma’ and ‘Persuasion’, both by Jane Austen = 0.99
- ‘Emma’ by Austen and ‘Julius Caesar’ by Shakespeare = 0.97
- ‘Emma’ by Austen and ‘Firefox’ from Webtext corpus = 0.86

Extraction – Discovery

Capturing patterns to discover context and use

Define a pattern

```
pattern = [{'LOWER': 'like’},

            {'LOWER': 'a’},

            {'POS': 'NOUN’}]
```

Returns

like a look
like a merit
like a gentleman
like a job
like a woman
like a bride
like a brother
like a daughter

UK Data Service

Extraction – Discovery

A more complex pattern

Define a pattern

```
pattern2 = [{'POS':'VERB’},
            {'LOWER': 'like’},
            {'LOWER': 'a’},

            {'DEP':'amod', 'OP':"?"},

            {'DEP':'amod', 'OP':"?"},

            {'DEP':'amod', 'OP':"?"},
            {'POS': 'NOUN'}]
```

Returns

looked like a sensible young man
argued like a young man
appear like a bride
seemed like a perfect cure
enters like a brother
writes like a sensible man

UK Data Service

Links to code, python packages and resources

- https://github.com/UKDataServiceOpen/text-mining/tree/master/code
- nltk (Natural Language Toolkit) https://www.nltk.org/book/ch01.html
- nltk.corpus http://www.nltk.org/howto/corpus.html
- spaCy https://nlpforhackers.io/complete-guide-to-spacy/
- Semantic vectors package https://github.com/semanticvectors/semanticvectors/wiki
- Geometry and Meaning, by Dominic Widdows https://web.stanford.edu/group/cslipublications/cslipublications/site/1575864487.shtml

Questions

Dr. J. Kasmire

julia.kasmire@manchester.ac.uk

@JKasmireComplex

UKDS

@UKDataService

UKDataService