LANGUAGE & AI: COLLECTING DATA

Chris Emmery
Department of Cognitive Science & AI
Tilburg University

@cmry • @_cmry • @cmry • cmry.github.io

⛏ COLLECTING LANGUAGE / TEXT

♻️ RECAP LAST LECTURE

- We looked at some interesting language **features**.
- We discussed some rudimentary ways of converting documents into **vectors**.
- Using this space, we did some **similarity** calculations.

PIPELINE

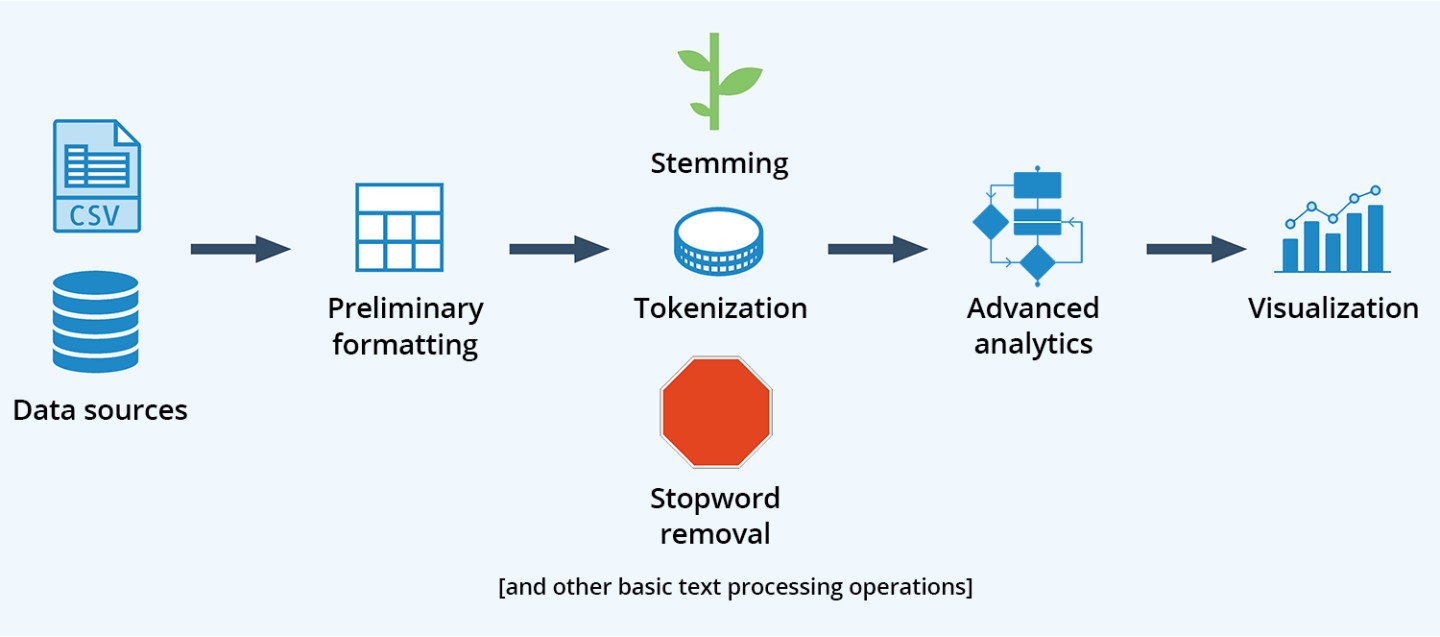


Image by Daniel Harris.

🖥️ WHAT DOES TEXT DATA LOOK LIKE?

```
<div data-test-id="post-content">
    <div class="_23h0-EcaBUorIHC-JZyh6J" style="width: 40px; border-left: 4px s
        <div class="_1E9mcoVn4MYnuBQSVDt1gC" id="vote-arrows-t3_apmsqk">
            <button aria-label="upvote" aria-pressed="false" class="voteButton
                <span class="_2q7IQ0BUOWeEZoeAxN555e _3SUsITjKNQ7Tp0Wi2jGxIM qW
            </button>
            <div class="_1rZYMD_4xY3gRcSS3p8ODO _3a2ZHWaih05DgAOtvu6cIo " style
                116k
            </div>
            <button aria-label="downvote" aria-pressed="false" class="voteButto
                <span class="_1iKd82bq_nqObFvSH1iC_Q Q0BxYHtCOJ_rNSPJMU2Y7 _2fe
            </button>
        </div>
    </div>
    <div class="_14-YvdFiW5iVvfe5wdgmET">
        <div class="_2dr_3pZUCk8KfJ-x0txT_1">
            <a data-click-id="subreddit" class="_3ryJoIoycVkA88fy40qNJc" href="
                <img style="background-color: rgb(87, 196, 199);" alt="Subreddi
            </a>
        </div>
        <div class="cZPZhMe-UCZ8htPodMyJ5">
            <div class="_3ASt
```

🗞️ APIS & DUMPS

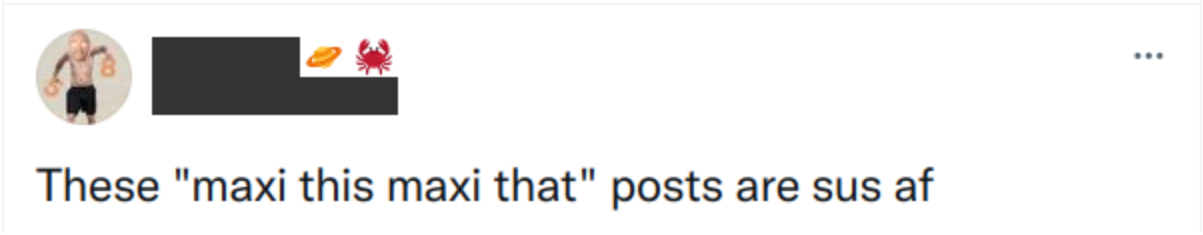
- 🤗 Datasets
- Twitter
- Wikipedia
- Reddit

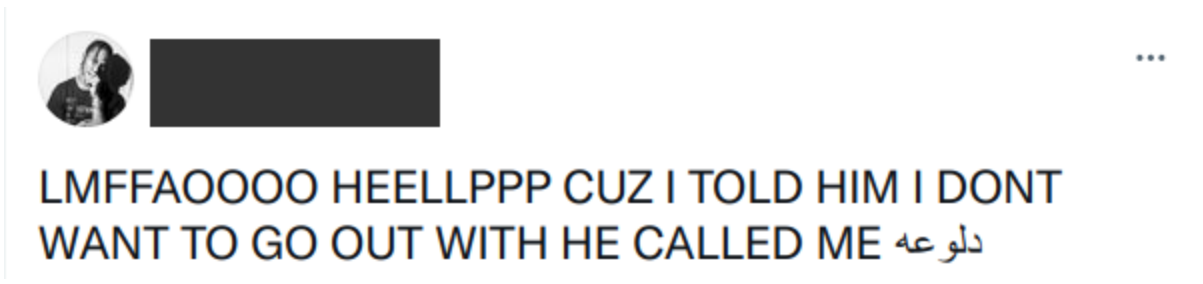
🗃️ STRUCTURED OBJECTS

e.g., JSON, XML


```
{
    "data": [
        {
            "id": "1212092628029698048",
            "text": "We believe the best future version of our API will come fr
            "possibly_sensitive": false,
            "referenced_tweets": [
                {
                    "type": "replied_to",
                    "id": "1212092627178287104"
                }
            ],
            "entities": {
                "urls": [
                    {
                        "start": 222,
                        "end": 245,
                        "url": "https://t.co/yvxdK6aOo2",
                        "expanded_url": "https://twitter.com/LovesNandos/status
                        "display_url": "pic.twitter.com/yvxdK6aOo2"
                    }
                ]
```

DOWN TO THE TEXT





🗑️ GARBAGE IN, GARBAGE OUT

- Can apply to many levels: data collection, sampling, sanitization, etc.
- For language: user-generated text is creative, which makes it a nightmare to work with.
- **Consider**: typos have a tremendous effect on the size of your vocabulary, and the representation of your documents (thus the similarity quality).

LANGUAGE VARIATION

- abbreviations, acronyms
- capitalization
- character flooding
- concatenations
- emoticons
- dialect, slang
- typos

✍️ FINDING & FIXING ERRORS

 REGULAR EXPRESSIONS

- A language to define string patterns.
- Available in many programming languages (in Python `re`).
- Can not only be used to **find** patterns but also to **replace** them.

```
import re
text = "You there, did you what you were looking for?"
patt = re.compile("you")
for match in patt.finditer(text):
    print(match)
```

```
<re.match object; span=(15, 18), match='you'>
<re.match object; span=(24, 27), match='you'>
```

🔀 REGEX: DISJUNCTIONS

this OR that

```
text = "You there, did you what you were looking for?"
regex_find("[Yy]ou", text)
```

```
<re.match object; span=(0, 3), match='You'>
<re.match object; span=(15, 18), match='you'>
<re.match object; span=(24, 27), match='you'>
```

🔀 REGEX: DISJUNCTIONS

```
text = "You there, did you what you were looking for?"
regex_find("[Yyou]", text)
```

```
<re.match object; span=(0, 1), match='Y'>
<re.match object; span=(1, 2), match='o'>
<re.match object; span=(2, 3), match='u'>
<re.match object; span=(15, 16), match='y'>
<re.match object; span=(16, 17), match='o'>
<re.match object; span=(17, 18), match='u'>
<re.match object; span=(24, 25), match='y'>
<re.match object; span=(25, 26), match='o'>
<re.match object; span=(26, 27), match='u'>
<re.match object; span=(34, 35), match='o'>
<re.match object; span=(35, 36), match='o'>
<re.match object; span=(42, 43), match='o'>
```

❌ REGEX: NEGATION

```
text = "You there, did you what you were looking for?"
regex_find("[^a-z]", text)
```

```
<re.match object; span=(0, 1), match='Y'>
<re.match object; span=(3, 4), match=' '>
<re.match object; span=(9, 10), match=','>
<re.match object; span=(10, 11), match=' '>
<re.match object; span=(14, 15), match=' '>
<re.match object; span=(18, 19), match=' '>
<re.match object; span=(23, 24), match=' '>
<re.match object; span=(27, 28), match=' '>
<re.match object; span=(32, 33), match=' '>
<re.match object; span=(40, 41), match=' '>
<re.match object; span=(44, 45), match='?'>
```

⭐ REGEX: KLEENE EXPRESSIONS

⭐ REGEX: KLEENE EXPRESSIONS

```
text = "You there, did you what you were looking for?"
regex_find("o[a-z]*", text)
```

```
<re.match object; span=(1, 3), match='ou'>
<re.match object; span=(16, 18), match='ou'>
<re.match object; span=(25, 27), match='ou'>
<re.match object; span=(34, 40), match='ooking'>
<re.match object; span=(42, 44), match='or'>
```

⭐ REGEX: KLEENE EXPRESSIONS

```
text = "You there, did you what you were looking for?"
regex_find("u[a-z]+", text)
```

⭐ REGEX: KLEENE EXPRESSIONS

```
text = "You there, did you what you were looking for?"
regex_find(".e.", text)
```

```
<re.match object; span=(5, 8), match='her'>
<re.match object; span=(28, 31), match='wer'>
```

*Matches are **greedy** by default.*

🔁 SUBSTITUTION

```
re.sub('!+', ' !', "Amazing!!!!!!!!!!")
```

```
'Amazing !'
```

More practice: https://regexr.com/

🤖 ELIZA: ROGERIAN THERAPIST

- Men are all alike.
  - IN WHAT WAY
- They're always bugging us about something or other.
  - CAN YOU THINK OF A SPECIFIC EXAMPLE
- Well, my boyfriend made me come here.
  - YOUR BOYFRIEND MADE YOU COME HERE
- He says I'm sad much of the time.
  - I AM SORRY TO HEAR YOU ARE SAD

🤖 ELIZA: JUST ROGERIAN REGEX

```
s/.* I'M (anxious|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (anxious|sad) .*/WHY DO YOU THINK YOU ARE \1?/
s/.* all .*/IN WHAT WAY?/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/
```

🥊 SYSTEM EVALUATION

🧙 LABEL, SEQUENCE, RANKING?

- $f(d) \rightarrow y$
- $f(d_t) \rightarrow y_t$
- $f(d; q) \rightarrow r$

🔨 BUILDING REGEX

Find all instances of "the".

🔨 BUILDING REGEX

Find all instances of "the".

the

🔨 BUILDING REGEX

Find all instances of "the".

the

Misses caps.

🔨 BUILDING REGEX

Find all instances of "the".

the

Misses caps.

[tT]he

🔨 BUILDING REGEX

Find all instances of "the".

the

Misses caps.

[tT]he

Matches words like 'there'.

🔨 BUILDING REGEX

Find all instances of "the".

the

Misses caps.

[tT]he

Matches words like 'there'.

[^a-zA-Z][tT]he[^a-zA-Z]

🔎 EVALUATION

Two error types:

- Matching ones that we shouldn't have (there, then): **false positives**.
- Not matching ones that we should have (The): **false negative**.

> *NLP / ML often face this as antagonistic goals to optimization.*

⁉️ CONFUSION MATRICES

	$\hat{y} = 1$	$\hat{y} = 0$
$y = 1$	TP	FN
$y = 0$	FP	TN

📊 OTHER METRICS

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_\beta \text{ score} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP}$$

📚 INFORMATION RETRIEVAL

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

Where $|Q|$ is the total number of queries, and rank_i the rank of the 1st relevant result.

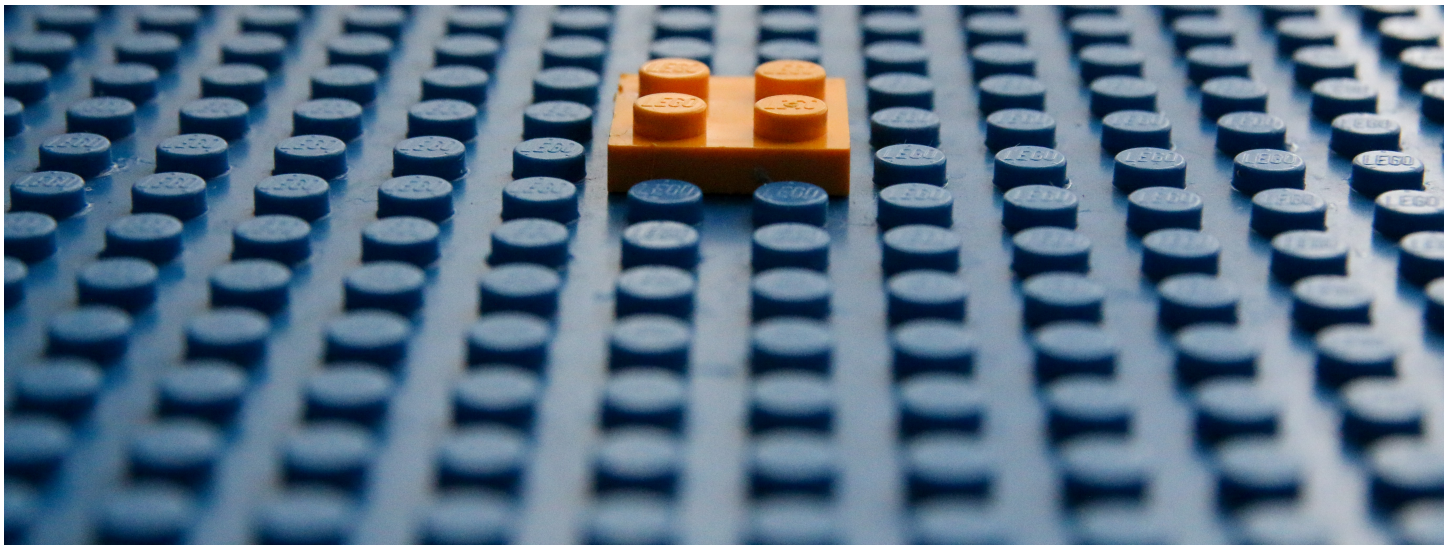
$$AP = \frac{\sum_{k=1}^{n} (P(k) * rel(k))}{\text{number of relevant items}}$$

Where $rel(k)$ is an indicator function which is 1 when the item at rank K is relevant, and $P(k)$ is the Precision@k metric.

🏷️ ANNOTATION

- Collect data.
- Have humans **annotate** labels, relevance, etc.
- Evaluate the **inter-rater agreement** (with e.g. Cohen's κ).

👮 NORMALIZATION



✨ VARIATION IN VIEW

Corpus	Tokens (N)	Types ($\lvert V \rvert$)
Shakespeare	884 K	31 K
Brown corpus	1 M	38 K
Switchboard	2.4 M	20 K
COCA	440 M	2 M
Google n-gram	1 T	13 M

Heap's Law: $|V| = k \cdot N^{\beta}$ *where* $0 < \beta < 1$ *(typically .67-75).*

🔨 COMMON APPROACHES

- **Case folding**: The → the.
- **Lemmatization**: He is reading sci-fi stories → He be read sci-fi story.
- **Stemming**: cats → cat, accurate → accur, copy → copi (Porter stemmer).

✅ CORRECTION: LEVENSHTEIN EDIT DISTANCE

```
i n t e * n t i o n
| | | | | | | | | |
* e x e c u t i o n

d s s   i s
```

```
def min_edit_dist(source, target):
    n, m = len(source), len(target)
    D[n+1,m+1] = zeroes

    for i in range(1, n):
        D[i, 0] = D[i-1, 0] + del_cost(source[i])
    for j in range(1, m):
        D[0, j] = D[0, j-1] + ins_cost(target[j])

    for i in range(1, n):
        for j in range(1, m):
            d[i, j] = min(D[i-1, j] + del_cost(source[i]),
                          D[i-1, j-1] + sub_cost(source[i], target[j])
                          D[i,   j-1] + ins_cost(target[i])
    return D
```

📍 DISTANCES + TRACEBACK

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

🖥️ ENCODINGS

- n-gram encoding (more next lecture).
- Byte-Pair Encoding (BPE, count-based merges).
- WordPiece Encoding (currently common, probability-based merges).

🙋 QUESTIONS

Post on the Discussion board and join class on Thursdays!