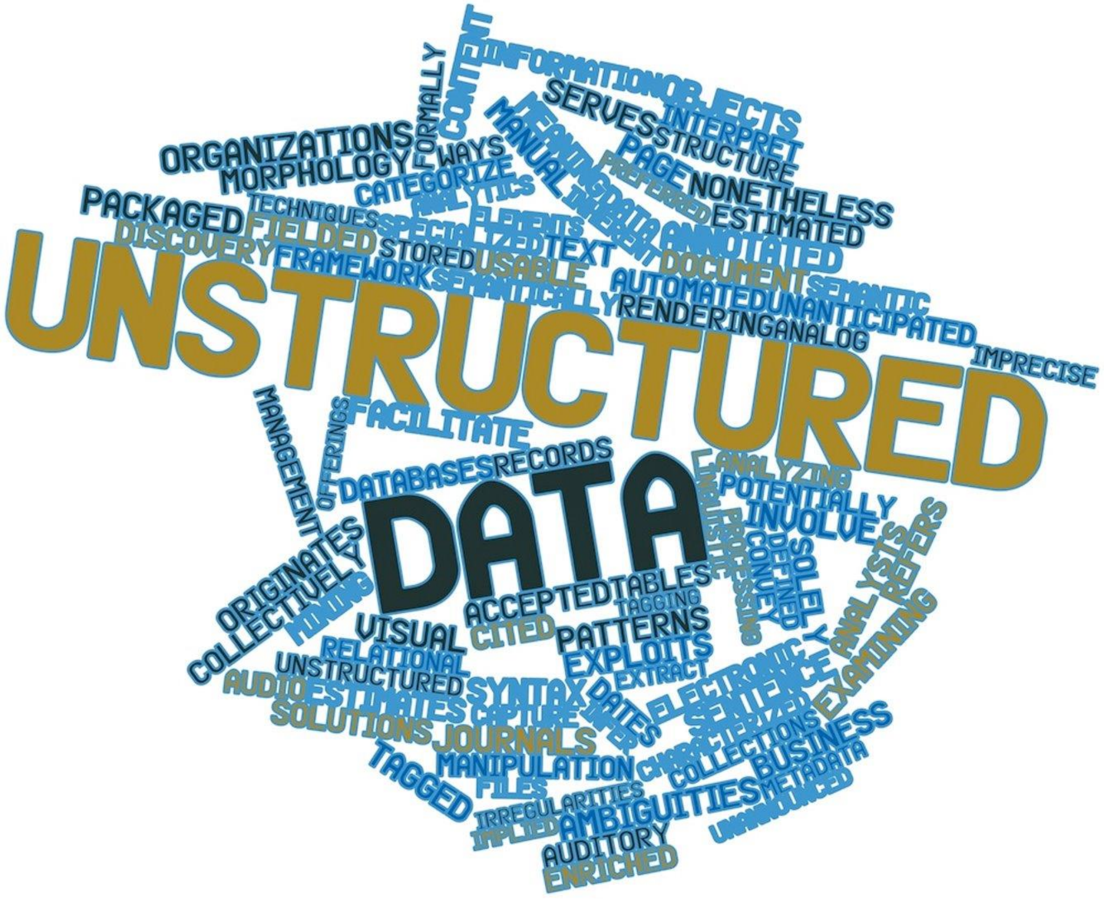
Text mining with Python and spaCy

Marcel Haas @DSC, Oct 2021

Text is unstructured data



While machine learning likes:

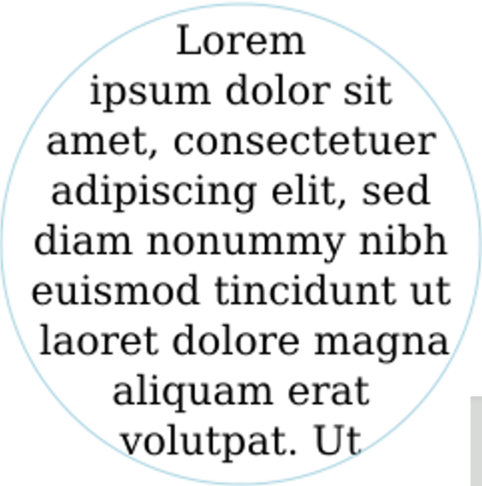
Table of baby-name data

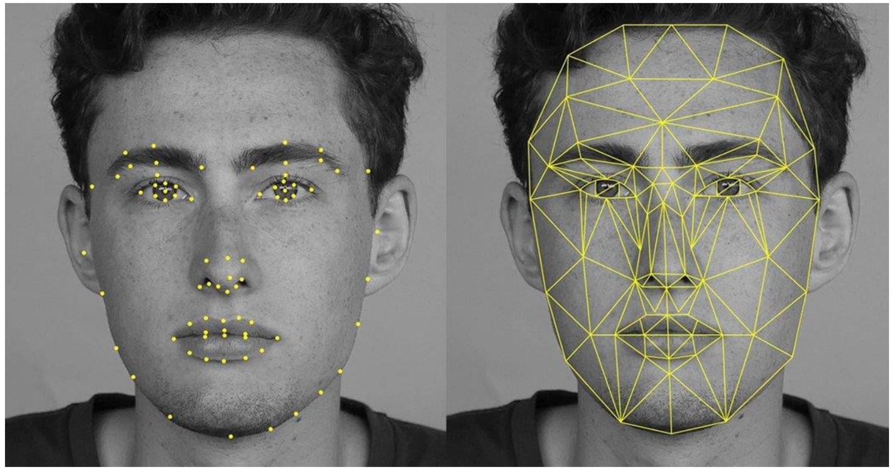
name	rank	gender	year
Jacob	1	boy	2009
Isabella	1	girl	2009
Ethan	2	boy	2009
Emma	2	girl	2009
Michael	3	boy	2009

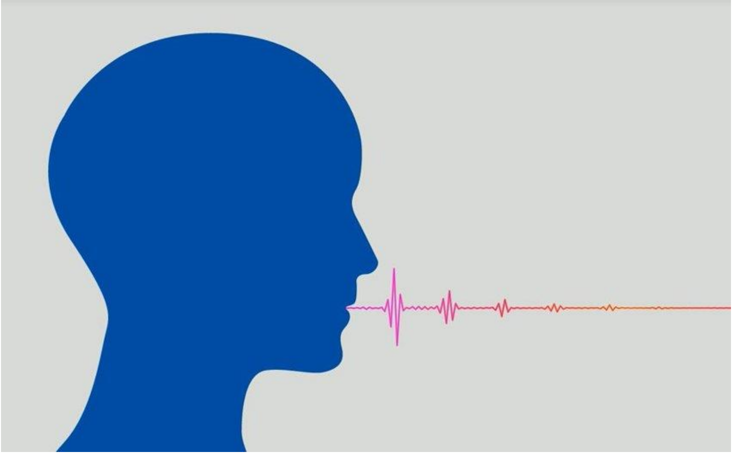
Field names

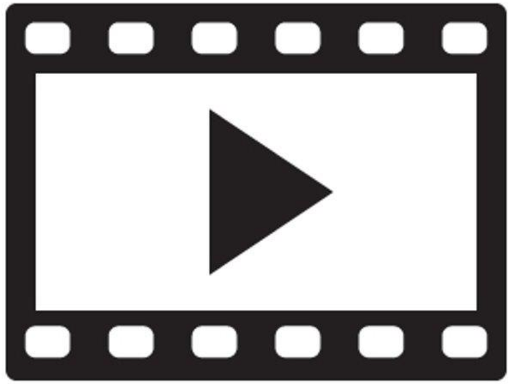
One row (4 fields)

More (un)structured data









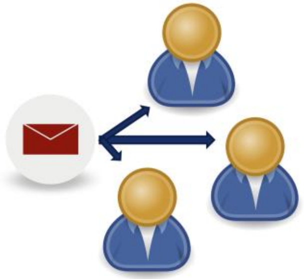
"The Data Science Approach"

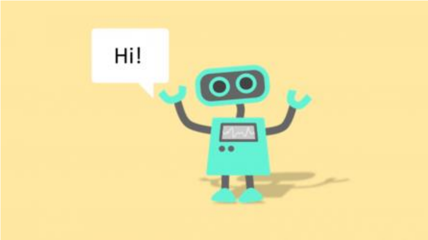
- Building blocks:
  - Text: words, n-grams, sentences
  - Other: pixels, patterns, shapes, …
- Tools to extract information from unstructured data
  - Structure in text
  - Grammar
  - Knowledge base

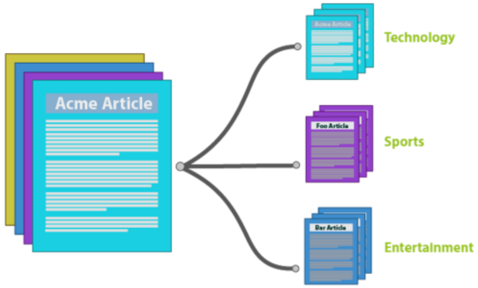
"Fdq brx uhdg wklv?“

"Can you read this?"
(in Ceasar's cypher)

What did I do in practice?







Text data in Python

```
[1]: import regex as re
     import string
     pattern = '[a-c]'
     sub = 'XXX'
     re.sub(pattern, sub, string.ascii_lowercase)

[1]: 'XXXXXXXXXdefghijklmnopqrstuvwxyz'
```

```
[2]: pattern = '\d{4}[ ]?[A-Z|a-z]{2}'
     sentence = "There is a Dutch postal code here 1234 AB, too! Two: 3729iu."
     re.findall(pattern, sentence)

[2]: ['1234 AB', '3729iu']
```

Data processing and cleaning

"The boy's 5 cars have different colours!"

- **Tokenization** - split the document or string into its constituents

[The, boy's, 5, cars, have, different, colours!]

- Or

[The, boy, s, 5, cars, have, different, colours, !]

Data processing and cleaning

[The, boy’s, 5, cars, have, different, colours!]

- Remove **punctuation** and other regex rules

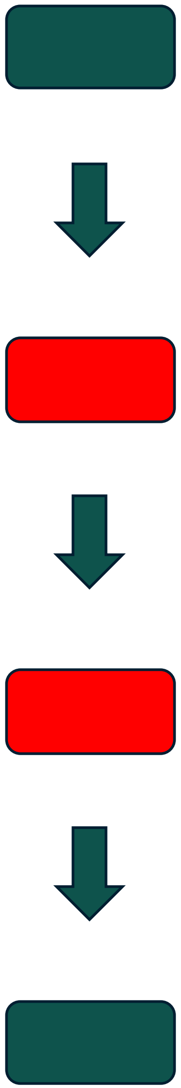
[The, boys, 000, cars, have, different, colours]

- Convert to **lowercase**

[the, boys, 000, cars, have, different, colours]

- Apply **synonym list** – (double) negatives, slang, abbreviations, etc

[the, boys, 000, cars, have, different, colors]



Data processing and cleaning

[the, boys, 000, cars, have, different, colors]

- Remove **stopwords**

[boys, 000, cars, different, colors]

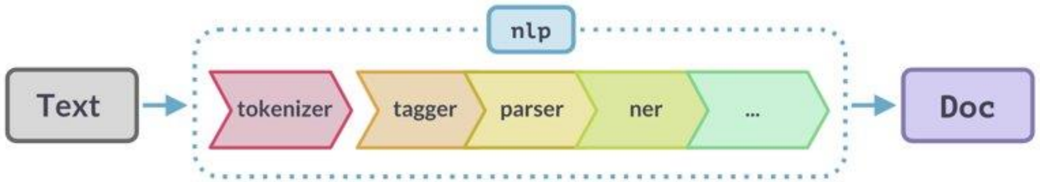
- **Stemming** or **lemmatization**

[boy, 000, car, differ, color]

All that and more: spaCy !

- Pre-trained language models (perhaps more in-depth after deep-learning sessions?)

- https://spacy.io has all info you need.



PROCESSING PIPELINE

NAME	COMPONENT	CREATES	DESCRIPTION
tokenizer	`Tokenizer`	`Doc`	Segment text into tokens.
tagger	`Tagger`	`Token.tag`	Assign part-of-speech tags.
parser	`DependencyParser`	`Token.head`, `Token.dep`, `Doc.sents`, `Doc.noun_chunks`	Assign dependency labels.
ner	`EntityRecognizer`	`Doc.ents`, `Token.ent_iob`, `Token.ent_type`	Detect and label named entities.
lemmatizer	`Lemmatizer`	`Token.lemma`	Assign base forms.
textcat	`TextCategorizer`	`Doc.cats`	Assign document labels.
custom	custom components	`Doc._.xxx`, `Token._.xxx`, `Span._.xxx`	Assign custom attributes, methods or properties.

Learning on text data: The Document-Term matrix

	boy	000	car	differ	color
Doc1					
Doc2					
Doc3					
Doc4					
Doc5					
Doc6					

- One **column** for each **term** (word or n-gram)
- One **row** for each **document**

Learning on text data: The Document-Term matrix

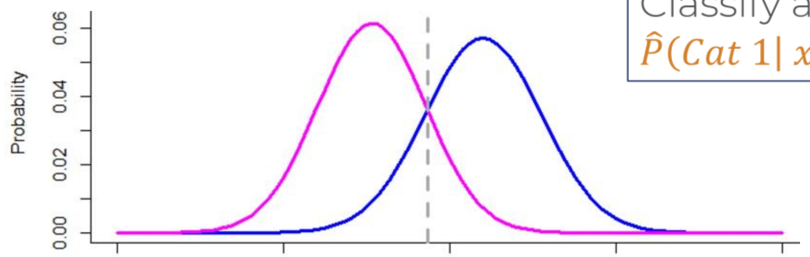
	boy	000	car	differ	color
Doc1	1.4	0	0.667	1.5	0
Doc2	0	0	0	0.75	1.9
Doc3	2.8	0.8	1	1.5	0
Doc4	0	1.6	0	0	1.9
Doc5	2.8	0.8	0	0	0
Doc6	5.6	0	0	0	0

- **TRUE / FALSE** – does this word occur in the document?
- **COUNT** – how many times does this word occur in the document?
- **TF IDF** – 'term frequency inverse document frequency' reflects how important this word is to the document by correcting for how often it occurs in the whole corpus.

Supervised: text classification

- Use Document term matrix as feature matrix for a supervised learning problem

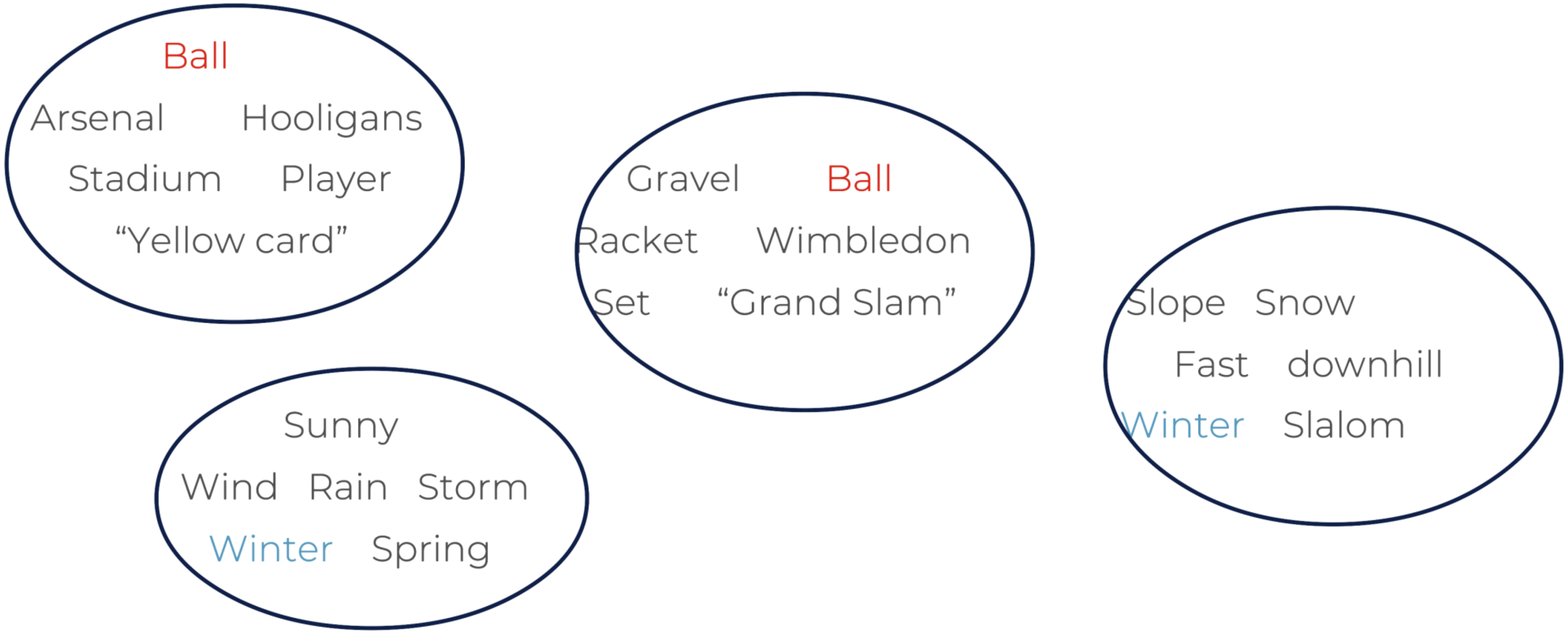
- Naive Bayes is a simple, but often quite effective algorithm

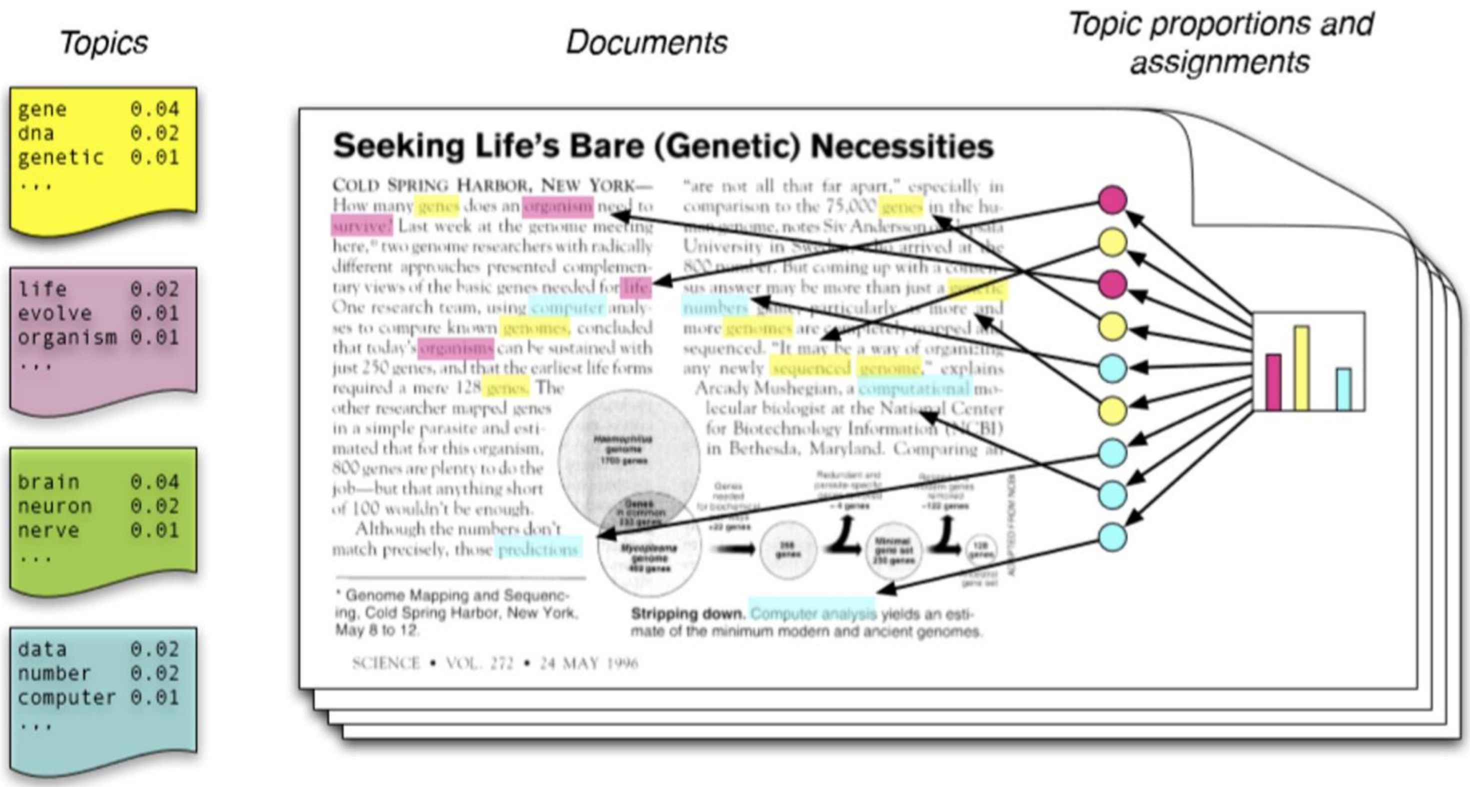


Feature word tf-idf

Classify as Category 1 if
$\hat{P}(Cat\ 1|\ x) > \hat{P}(Cat\ 2\ |\ x)$

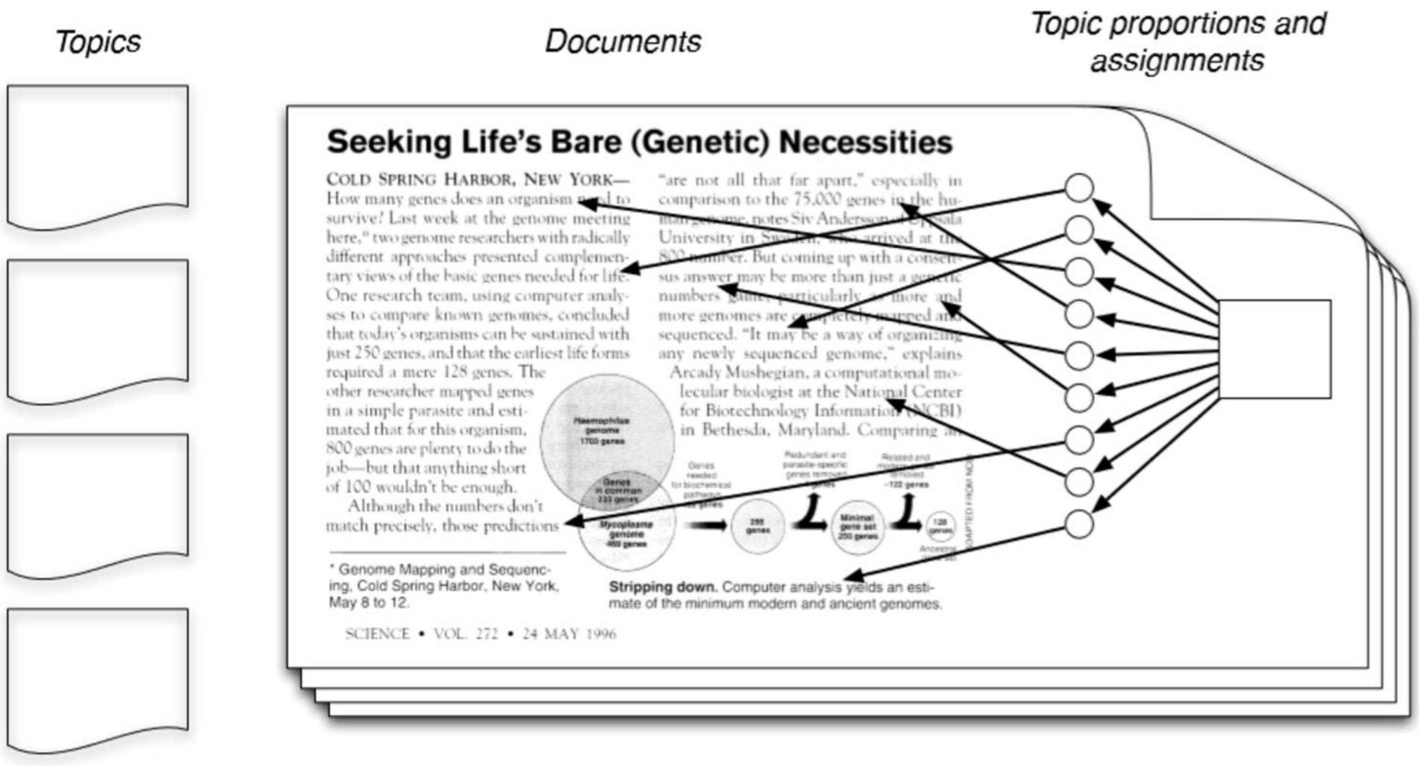
Unsupervised: topic modeling





- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

http://www.cs.columbia.edu/~blei/talks/Blei_MLSS_2012.pdf



- In reality, we only observe the documents
- The other structure are **hidden variables**
- Topic modeling algorithms **infer** these variables from data.

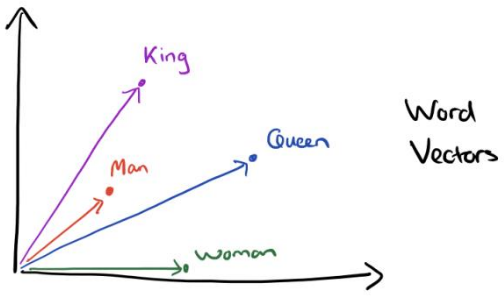
Latent Dirichlet Allocation

- Randomly assign each word in each document to one of the K topics.
- For each document d...
  - For each word w in d...
    - And for each topic t, compute two things:
      - 1) p(topic t | document d) = the proportion of words in document d that are currently assigned to topic t, and
      - 2) p(word w | topic t) = the proportion of assignments to topic t over all documents that come from this word w.
    - Reassign w a new topic t with probability p(topic t | document d) * p(word w | topic t)
- And repeat...

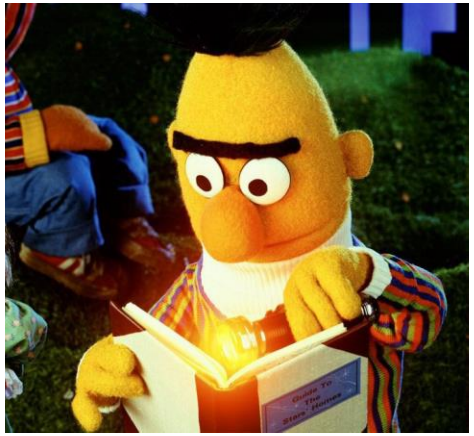
We're assuming that all topic assignments except for the current word in question are correct, and then updating the assignment of the current word using our model of how documents are generated.

Moving on: terms in their proper context

- Word vectors and document vectors



- Neural net-based language models (CNN, RNN, Transformers)
  - Pre-trained
  - Re-trained for domain adaptation



- …

I'll stop talking

- In the notebook, these topics come by:

https://github.com/harcel/TextMiningWorkshop

- There are exercises, for which you can load *example* solutions.

- In the afternoon: let's get lost in your own text data!