NLP and Preprocessing

Raf Alvarado
UVA DS 5001

Data models, NLTK, NLP annotation,
POS prediction with HMMs

Business

Extension for HW03: Tomorrow midnight

Lab Review

Computing NGram Models revisions

- NGram Data **Representation**
- NGram Count **Smoothing**

token_num	w0
0	<s>
1	this
2	was
3	the
4	page
5	at
6	which
7	the
8	favourite
9	volume
10	always
11	opened
12	</s>

token_num	w0	w1
0	</s>	<s>
1	<s>	this
2	this	was
3	was	the
4	the	page
5	page	at
6	at	which
7	which	the
8	the	favourite
9	favourite	volume
10	volume	always
11	always	opened
12	opened	</s>

token_num	w0	w1	w2
0	</s>	</s>	<s>
1	</s>	<s>	this
2	<s>	this	was
3	this	was	the
4	was	the	page
5	the	page	at
6	page	at	which
7	at	which	the
8	which	the	favourite
9	the	favourite	volume
10	favourite	volume	always
11	volume	always	opened
12	always	opened	</s>

The idea is to **preserve the target sentence representation** across ngram histories.

As opposed to this →

	w0	w1	w2
0	<s>	the	family
1	the	family	of
2	family	of	dashwood
3	of	dashwood	had
4	dashwood	had	long

```
pd.concat([x.shift(i) for i in range(i,-1,-1)], axis=1)
```

To build this representation, you need to work backwards when binding the padded token sequence x.

|V|

w0			1	1760	1784	1800	1810	1814	200	29th	...
w1	1760	1785	ends	married	elizabeth	he	charles	wearing	<UNK>	of	...
w2											
1	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	...
1760	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	...
1784	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	...
1785	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	...
1787	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	13.023581	...
...	...	...	...	...	...	...	...	...	...	...	...

To compute conditional probabilities, we smooth e.g. by |V| in the denominator not $|V^2|$ for bigrams

```python
def ngrams_to_models(ngrams, k=.01):

    model = [None for i in range(ngram_order)]
    K = len(VOCAB2) * k

    for i in range(ngram_order):
        if i == 0:
            model[i] = ngrams[i].value_counts().to_frame('n')
            model[i]['p'] = (model[i].n + k) / (model[i].n.sum() + K)
            model[i]['i'] = np.log2(1/model[i].p)
        else:
            model[i] = ngrams[i].value_counts().to_frame('n')
            model[i]['cp'] = (model[i].n + k) / (model[i-1].n + K)
            model[i]['i'] = np.log2(1/model[i].cp)
        model[i] = model[i].sort_index()

    return model
```

Review

So far, we have learned about two kinds of **models**:

- **Text** models, which describe:
  - Structure and function of text as text (**discourse**)
  - **OHCO**
  - Text as **message** (symbol sets and sequences)
- **Language** models, which describe:
  - Latent generative **grammar** in information source
  - Modeled as an **n-gram** probability distribution
  - **Inferred** from available texts
  - Used to **predict** and **generate** sentences

Review

In this lesson, we **consolidate** these models into **a single data model**

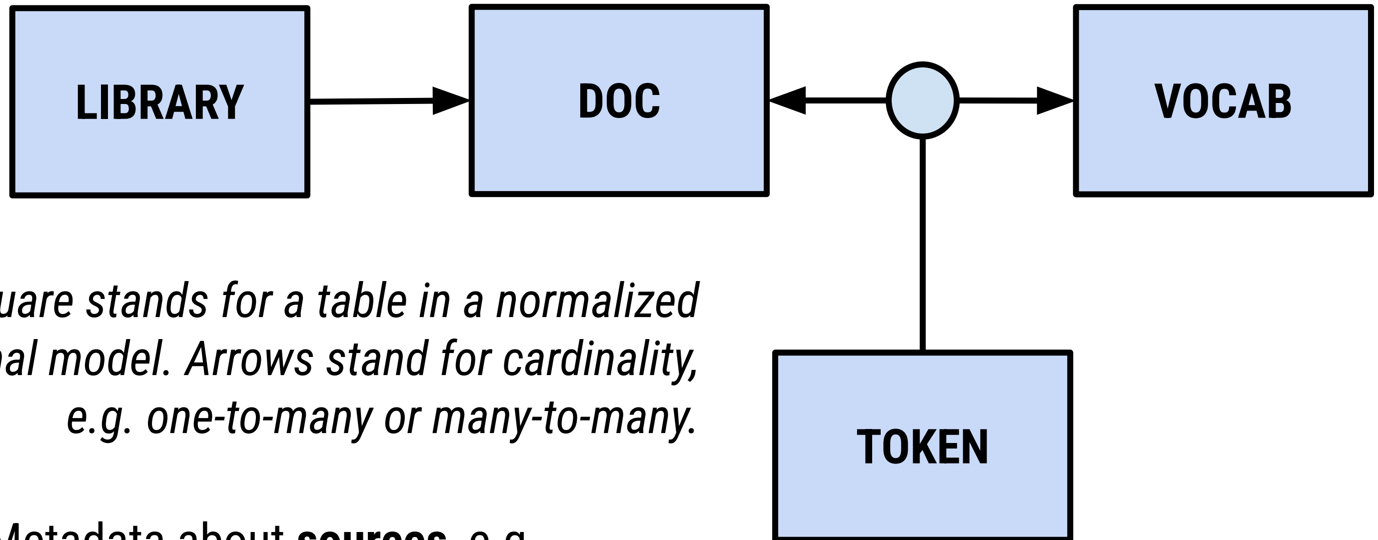
- A simple **relational** model that you can implement in **SQL**
- Or simply in CSV and data frames

We have already been developing this model **implicitly** in our notebooks with **Pandas**

- Source **documents imported and converted** into
  - **Vocabulary** dataframe
  - **Tokens** dataframe

Let's look at this more **formally** ...

The Data Model — Core Tables



Each square stands for a table in a normalized relational model. Arrows stand for cardinality, e.g. one-to-many or many-to-many.

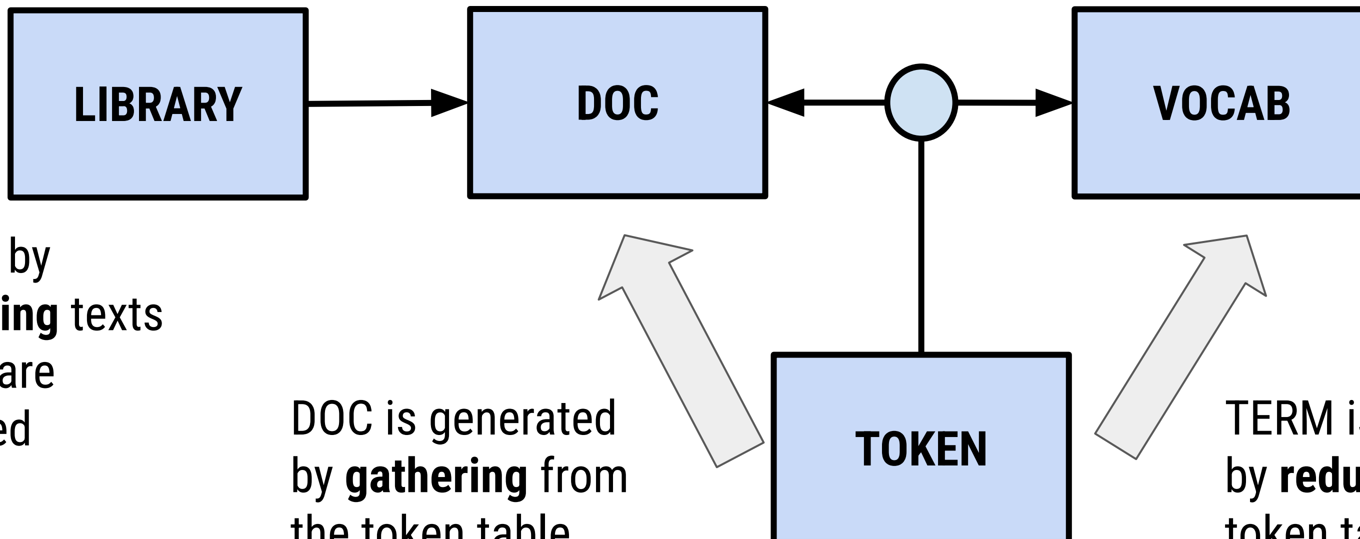
LIBRARY: Metadata about **sources**, e.g. books, essays, etc. Publication date, author, etc. Often omitted, but contains labels.

DOC: Discursive units parsed from sources, e.g. chapters or paragraphs. Called a **corpus** of documents in the analytic literature. These are **messages** in Shannon's model. Often expressed as an **OHCO** in the **TOKEN** index.Called "**docs**" and "**corpus**" in NLP

VOCAB: The distinct set of terms (i.e. token types) that exist in the library. Often called a **vocabulary** or **dictionary**. The **symbol set** in Shannon's terms. Can be imported from external source for each language in the library.

TOKEN: The term instance **selected** at a particular point in the symbol sequence that composes the message.

The Data Model – Process of Generation



Created by **registering** texts as they are tokenized

DOC is generated by **gathering** from the token table

TERM is generated by **reducing** the token table

Generated by **chunking and parsing** source files

The Data Model – Some Fields

LIBRARY

ID, title, **date**, **author**, platform/publisher, etc.

DOC

OHCO levels*, label+, model attributes, gathered tokens

TOKEN

OHCO key, sequence num, string, term ID, NLP attributes, model attributes

VOCAB

ID, string, NLP attributes, statistical attributes, model attributes

* Levels are variable – may be chapters, paragraphs, etc.depending on application

Example VOCAB Table

term_id	term_str	n	p	port_stem	stop	df	idf	tfidf_sum
Filter	Filter	Fil...	Filter	Filter	Filter	Filter	Filter	Filter
54	abnormally	3	1.99944415...	abnorm	0	3	2.02802872...	6.08408617...
55	aboard	16	1.06637021...	aboard	0	12	1.42596873...	22.8154997...
56	abode	29	1.93279601...	abod	0	20	1.20411998...	34.9194794...
57	abodes	2	1.33296276...	abod	0	2	2.20411998...	4.40823996...
58	abominable	10	6.66481384...	abomin	0	8	1.60205999...	16.0205999...
59	abominably	3	1.99944415...	abomin	0	3	2.02802872...	6.08408617...
60	abomination	2	1.33296276...	abomin	0	2	2.20411998...	4.40823996...
61	aboon	2	1.33296276...	aboon	0	1	2.50514997...	5.01029995...
62	aboord	1	6.66481384...	aboord	0	1	2.50514997...	2.50514997...
63	aboot	3	1.99944415...	aboot	0	2	2.20411998...	6.61235994...
64	aborigines	1	6.66481384...	aborigin	0	1	2.50514997...	2.50514997...
65	abortion	1	6.66481384...	abort	0	1	2.50514997...	2.50514997...
66	abortive	4	2.66592553...	abort	0	4	1.90308998...	7.61235994...
67	abound	5	3.33240692...	abound	0	5	1.80617997...	9.03089986...
68	abounded	1	6.66481384...	abound	0	1	2.50514997...	2.50514997...
69	abounding	3	1.99944415...	abound	0	3	2.02802872...	6.08408617...
70	abounds	1	6.66481384...	abound	0	1	2.50514997...	2.50514997...
71	about	2000	0.00133296...	about	1	295	0.03532796...	70.6559246...
72	above	329	0.00021927...	abov	1	151	0.32617303...	107.310927...
73	aboveboard	1	6.66481384...	aboveboard	0	1	2.50514997...	2.50514997...
74	abraham	4	2.66592553...	abraham	0	3	2.02802872...	8.11211489...

Note addition of columns for **statistical** and **linguistic** features

These may be added throughout the pipeline

Example TOKEN Table

genre	author	book	chapter	para_num	sent_num	token_num	pos	token_str	punc	num	term_str	term_id
Filter	Filter	Filter	Fil...	Filter	Filter	Filter	Filter	Filter	F...	F...	Filter	Filter
d	christie	secretadvers...	1	0	0	0	.	.	1	0	NULL	-1
d	christie	secretadvers...	1	0	1	0	DT	THE	0	0	the	24127
d	christie	secretadvers...	1	0	1	1	NNP	YOUNG	0	0	young	27354
d	christie	secretadvers...	1	0	1	2	NNP	ADVENTURE...	0	0	adventurers	399
d	christie	secretadvers...	1	0	1	3	NNP	LTD.	0	0	ltd	14406
d	christie	secretadvers...	1	1	0	0	JJ	“TOMMY,	0	0	tommy	24529
d	christie	secretadvers...	1	1	0	1	JJ	old	0	0	old	16509
d	christie	secretadvers...	1	1	0	2	NN	thing!”	0	0	thing	24202
d	christie	secretadvers...	1	2	0	0	JJ	“Tuppence,	0	0	tuppence	25026
d	christie	secretadvers...	1	2	0	1	JJ	old	0	0	old	16509
d	christie	secretadvers...	1	2	0	2	NN	bean!”	0	0	bean	2000
d	christie	secretadvers...	1	3	0	0	DT	The	0	0	the	24127
d	christie	secretadvers...	1	3	0	1	CD	two	0	0	two	25109
d	christie	secretadvers...	1	3	0	2	JJ	young	0	0	young	27354
d	christie	secretadvers...	1	3	0	3	NNS	people	0	0	people	17404
d	christie	secretadvers...	1	3	0	4	VBN	greeted	0	0	greeted	10778
d	christie	secretadvers...	1	3	0	5	DT	each	0	0	each	7648

Note again presence of **statistical** and **linguistic** features
These get added over time

Principle: Everything has a place

Forms of Text Data

F0	**Source Format**. The initial source format of a text, which varies by collection, e.g. XML (e.g. TEI and RSS), HTML, plain text (e.g. Gutenberg), JSON, and CSV.	Locating archives, text encoding, character encoding, digital humanities collections development.
F1	**Machine Learning Corpus Format (MLCF)**. Ideally a table of minimum discursive units indexed by document content hierarchy.	Document models, OHCO vs stream models of text.
F2	**Standard Text Analytic Data Model (STADM)**. A normalized set of tables including DOC, TOKEN, and TERM tables. Produced by the tokenization of F1 data.	Tokenization methods, entity-relationship models of text, sentence segmentation.
F3	**NLP Annotated STADM**. STADM with annotations added to token and term records indicating stopwords, parts-of-speech, stems and lemmas, named entities, grammatical dependencies, sentiments, etc.	Natural language processing methods to infer annotations. Statistical vs regular expression methods.
F4	**STADM with Vector Space models**. Vector space representations of TOKEN data and resulting statistical data, such as term frequency and TFIDF.	Sparse and dense matrix representations,document-term matrices, frequency measures.
F5	**STADM with analytical models**. STADM with columns and tables added for outputs of fitting and transforming models with the data.	Non-deterministic machine learning models.
F6	**STADM converted into interactive visualization**. STADM represented as a database-driven application with interactive visualization, .e.g. Jupyter notebooks and web applications.	Visualization, user-interface design, data product design.

Machine Learning Corpus Format

doc_id	doc_label	doc_content
0015	en	In April 2019, an Israeli lunar lander, Beresh...
0017	fr	Le projet Afripédia a notamment fait usage de ...
0035	it	Wikipedia mantiene un approccio ottimistico su...
0026	ja	ラリー・サンガーはプロジェクトの発足から1年と数か月の間、「Bomis」から賃金の支払いを受...
0041	fr	Sur chaque page, des onglets permettent d'accé...
0040	pt	Em maio de 2014 havia 277 versões ativas da Wi...
0024	de	Im Dezember 2008 blockierten britische Provide...
0039	pl	Pomimo że istnieją (bądź istniały) inne projek...
0010	it	Da quando Wikipedia ha raggiunto un considerev...
0039	nl	In 2005 publiceerde Nature de resultaten van e...
0034	ja	多言語化に乗り出したのは2001年の5月頃であると思われる。当時の発表によれば12前後の非英...
0074	pt	Em julho de 2009, a BBC Radio 4 transmitiu uma...
0077	es	Hay contenidos que no tienen cabida en Wikiped...
0087	fr	Une description précise de l'architecture des ...
0107	en	Several languages of Wikipedia also maintain a...
0088	de	Das Wiki-Prinzip bezeichnet funktionale und ps...
0009	pt	Vários outros projetos de wiki-enciclopédias f...
0042	ja	2003年11月、ロシア語版ウィキペディアでライセンス形態についての論争が基となり、一部の利...
0016	pt	Em junho de 2010, os administradores anunciara...
0070	ja	他の、よりウィキペディアに近い活動として、自発的な参加によって新しくフリーな情報源を作り上げ...
0011	fr	Plusieurs autres moyens de consulter l'encyclo...
0010	en	In January 2007, Wikipedia entered for the fir...
0008	ja	2016年2月時点で約290の言語版の記事数の総計は3800万以上に上り、最も多い英語版が約...

Gensim
SKLearn
TextBlob
SpaCy
NLTK
MALLET

Deterministic and Non-deterministic Tables

Deterministic

Generated by specifiable rules

Same result each time

Non-deterministic

Generated by sampling methods. e.g. Gibbs Sampling in a topic model

Different results each time

Sensitive to hyperparameters

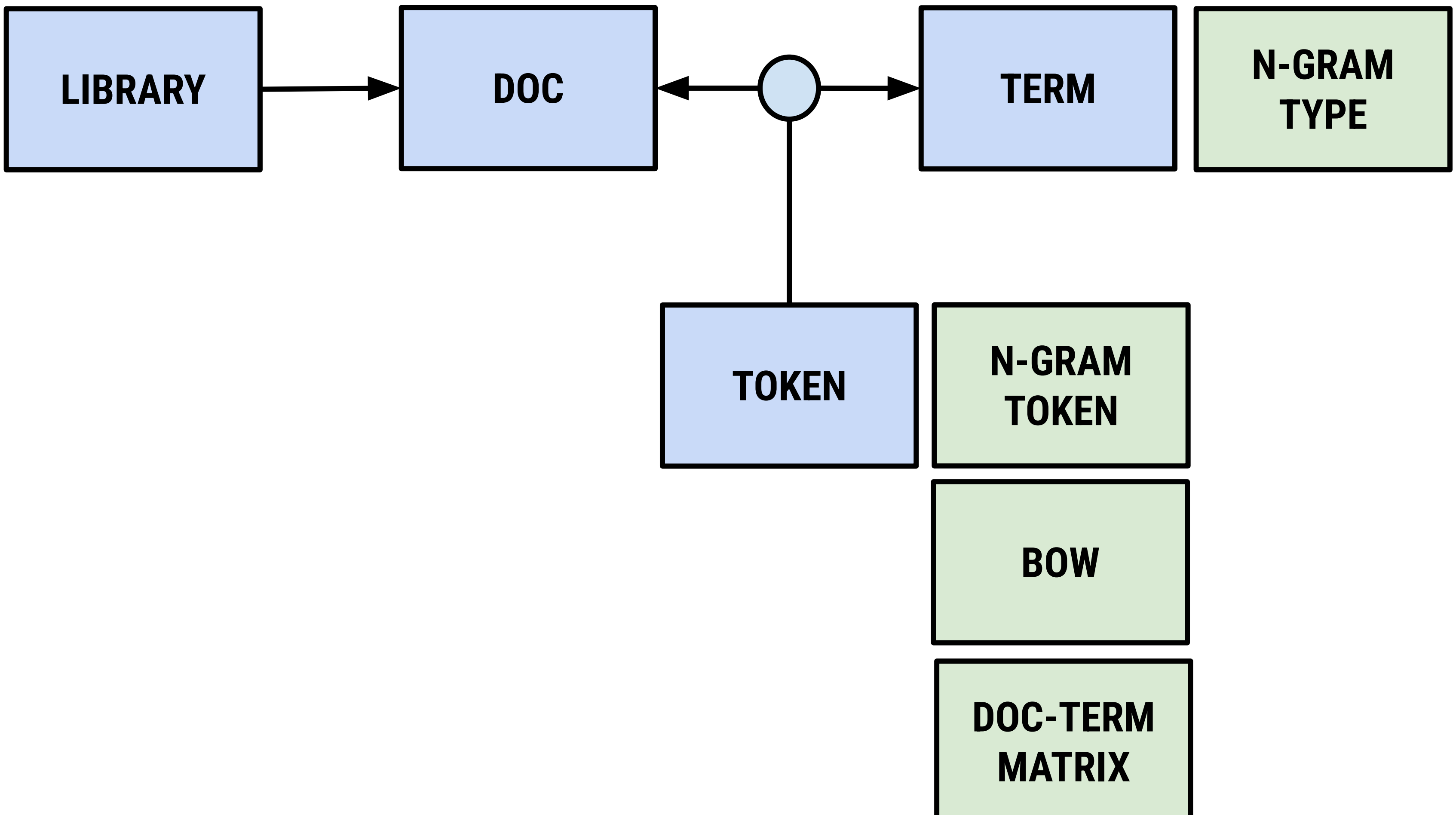
Digital Analytical Editions should provide these rules and parameters

Example Non-deterministic Table

word_str	0	1	2	3	4	5	6
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
aback	1.84541789...	0.04019420...	3.98538132...	-0.0201189...	0.07591186...	-0.0521244...	1.93987856...
abaft	4.11426599...	0.04886889...	3.83057170...	-0.0232496...	0.00642175...	-0.1592261...	3.77589417...
abandon	-1.5640388...	-0.0776159...	-9.0297566...	0.14930089...	-0.2128657...	0.07775350...	9.50455416...
abandoned	5.44989854...	-0.0675770...	-5.9952514...	0.06094020...	-0.2475086...	-0.0286770...	2.49075975...
abandoning	-1.1993801...	-0.0332486...	-2.2007837...	-0.0685863...	-0.0511088...	0.10835312...	-4.0959031...
abandons	-3.9300572...	0.01607787...	-3.7639176...	0.06245690...	0.08652514...	-0.0506616...	1.81168854...
abasement	-3.8413611...	-0.1042514...	-4.7824666...	-0.0383019...	0.14986044...	-0.0506771...	2.15020559...
abashed	-7.3309750...	0.02831955...	2.02956677...	-0.0531437...	-0.0072086...	-0.0530006...	1.01284059...
abate	3.03987752...	-0.0025216...	3.81302841...	-0.1152536...	0.09615959...	-0.1626534...	3.84900037...
abated	3.44784455...	0.02226876...	2.26976690...	-0.0005622...	-0.0097909...	-0.0173760...	4.09753279...
abatement	-8.2874177...	0.00632240...	-2.3095555...	-0.0131895...	0.09824863...	0.08774332...	-3.1235213...
abating	-4.4238080...	-0.0282851...	-4.3072031...	0.12425188...	0.00175382...	-0.1169347...	6.97296907...
abbess	-7.1392284...	0.01343215...	-2.5158886...	0.10948297...	-0.0744403...	-0.0773908...	2.94171784...

Detail of word embeddings for the vocabulary. These might be added as columns to the core Vocabulary table, or put in a separate table and joined by a key.

The Data Model – Added Tables



Digital Analytical Editions

The data model is the **foundation** for a digital analytical edition of a corpus

- Ideally, it is accompanied by a **data dictionary** describing each table and field

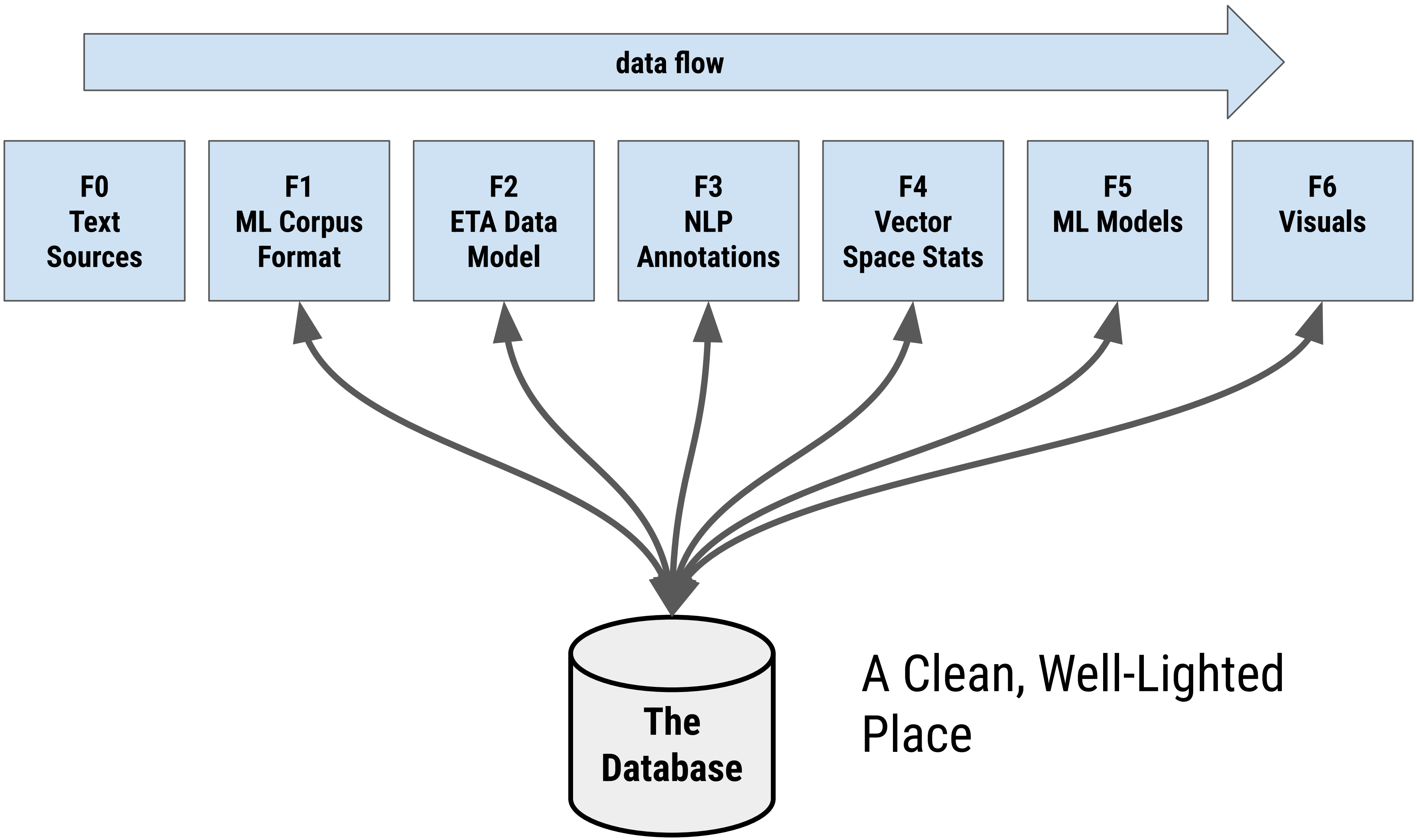
- Also should be documented for **parameters** used to generate tables

Non-deterministic tables are part of the edition, even though they are transient

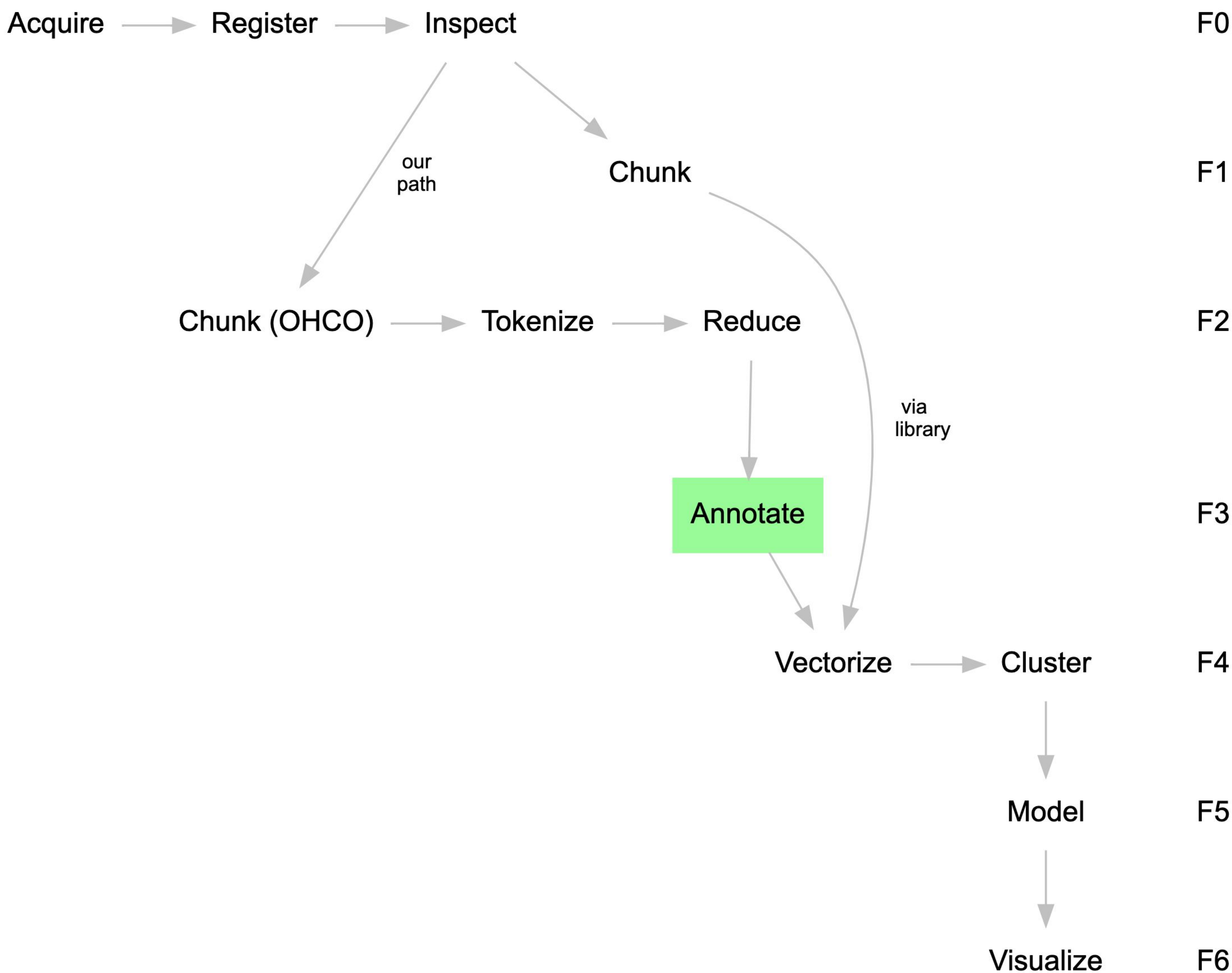
- You may generate hundreds of topic models, for example

- But the models you use for your results should be **preserved**

The Pipeline and the Database



A Clean, Well-Lighted Place

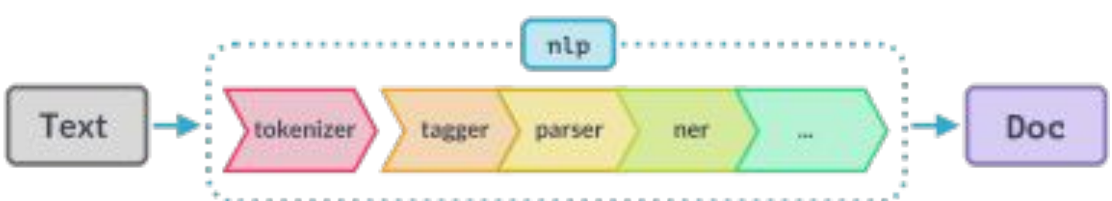

Acquire
Register
Inspect
F0
our path
Chunk
F1
Chunk (OHCO)
Tokenize
Reduce
F2
via library
Annotate
F3
Vectorize
Cluster
F4
Model
F5
Visualize
F6

spaCy

Here is sPacy's model …

See https://spacy.io/

NAME	DESCRIPTION
Doc	A container for accessing linguistic annotations.
DocBin	A collection of Doc objects for efficient binary serialization. Also u
Example	A collection of training annotations, containing two Doc objects: t predictions.
Language	Processing class that turns text into Doc objects. Different langua of it. The variable is typically called nlp .
Lexeme	An entry in the vocabulary. It's a word type with no context, as opp has no part-of-speech tag, dependency parse etc.
Span	A slice from a Doc object.
SpanGroup	A named collection of spans belonging to a Doc .
Token	An individual token — i.e. a word, punctuation symbol, whitespace,



Preprocessing with NLP

NLP Annotations

LMs are the **foundation** of NLP

They are used to **predict** and estimate a variety of **lexical features**

Lexical = word-level = tokens and terms (vocabulary)

In general, these features are called **annotations** and typically include:

Stop words ← (in my book)

Part of Speech

Lemmas (not = stemming)

Grammatical dependency

Named entities

Thesauri and dictionaries (e.g. sentiment)

Etc.

In general, we will **not** be using our own LMs to generate these annotations

Instead, **we use libraries** that provide these for us, such as those provided by NLTK or spaCy

See also Stanford NLP (Java)

About NLTK, the Natural Language Toolkit

Python library first released in 2001 by Stephen Bird and Edward Loper of **University of Pennsylvania linguistics**

- Documented by the "**NLTK book**," a standard reference in the field

Has **many libraries** to accomplish many tasks, but:

- Uses **older** Python conventions
- Documentation **not well organized**
- Being replaced by more scalable libraries, e.g. **spaCy**
- Supports **multiple languages** and **approaches**

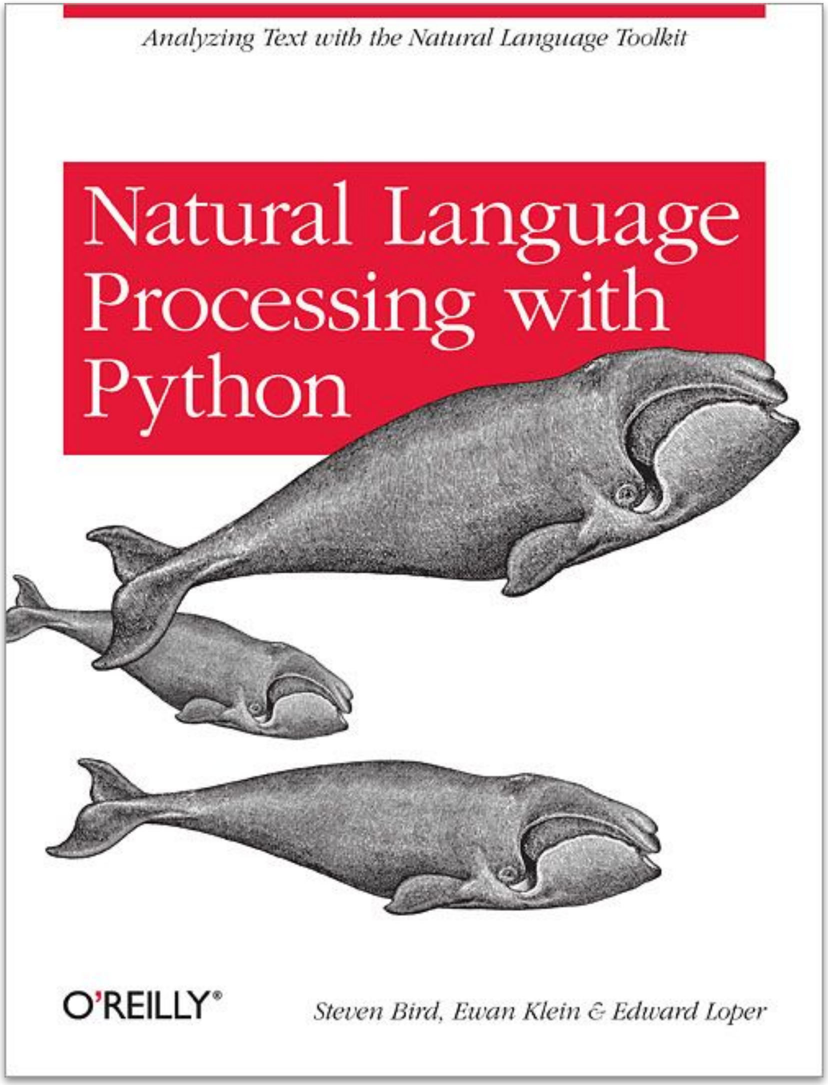
Still very useful!

NLTK Book

Book available, but only covers Python 2

New version online at http://www.nltk.org/book/

A good introduction to NLP, but lacks mathematical foundations



Linguistic Annotation and Our Model

Linguistic annotations are **features** added to the tables in our model

DOC: language, etc.
TOKEN: part-of-speech, grammatical dependency, etc.
TERM: stopword, frequency, etc.

Routine tasks like stopword removal and text normalization are not often considered annotations, but they really are

They are based on **linguistic inferences** codified in dictionaries and other resources

In this course, **we do not infer these** features (for the most part) but **borrow** libraries and apply them to our texts

Kinds of Annotations (more detail)

Orthographic features: identify and normalize case (shape), punctuation

Stop words: identify words that may be omitted from many tasks

Part of Speech (POS): Identify for each token, e.g. noun, verb, adjective, etc.

Lemmas and stemming: Group related words, e.g. *run* and *running*

Grammatical dependency: Identify syntactic structure of a sentence, represent as a tree

Named entities: Identify kinds of nouns, e.g. PERSON, PLACE, etc.

Also: Identify tokens in thesauri and dictionaries (e.g. **sentiment**)

NLP also Used for Segmentation Tasks

Tokenization: Extraction of tokens, e.g. words and punctuation; e.g. ability to distinguish uses of periods for abbreviation or sentence endings. AKA **parsing**.

Sentence Segmentation: Sentences and clauses can't be inferred from punctuation alone, e.g. Dr. and Mrs.

Paragraph Segmentation: Paragraphs and other structural elements (headings, chapters, etc.) may need to be explicitly annotated (as we have seen in identifying OHCO)

Text segmentation is sometimes called **chunking.**

Two Main Approaches

Pattern matching with **regular expressions** (“regex”)

Need to understand basic regular expressions!

NLP methods based on **language models** and other methods

NLP and computational linguistics have built **a body of knowledge** in the form of dictionaries and other resources

These are the results of sophisticated **algorithmic and scholarly approaches** to language

These results are encoded in program libraries and available through tools like **Python’s NLTK library**

Lemmas and Stems

Lemmatization is the process of **grouping** together the different inflected forms of a word so they can be analysed as a single item.

Comes from linguistics
Requires knowledge of context, provided by ngram LMs

Stemming is the process for **reducing** inflected or derived words to a stem, base or root form. The stem need not be identical to the morphological root of the word.

Developed for information retrieval
Does not require knowledge of context

Stemmers: Porter, Lancaster, Snowball

Stemming	Lemmatization
`adjustable` → `adjust`	`was` → `to be`
`formality` → `formalit`	`better` → `good`
`airliner` → `airlin`	`meeting` → `meeting`
Requires use of **rules**, eg. regular expressions	Require external **info**, e.g. grammar and dictionary

Segmentation and Tokenization

We have handled sentence segmentation and tokenization with regular expressions so far

- But we've run into **problems** – e.g. Mrs. being treated as end of sentence, inconsistent segmentation
- Also, we want to **preserve punctuation** in our data model so we can rebuild the text if necessary

So, we will use NLTK to help us with this

- NLTK has many **tokenizers**, some of them trainable on new corpora

NLTK Tokenization Example

s1 = "On a $50,000 mortgage of 30 years at 8 percent, the monthly payment would be $366.88."

s2 = "\"We beat some pretty good teams to get here,\" Slocum said."

```
nltk.word_tokenize(s1)

['On', 'a', '$', '50,000',
'mortgage', 'of', '30',
'years', 'at', '8',
'percent', ',', 'the',
'monthly', 'payment',
'would', 'be', '$', '366.88',
'.']
```

```
nltk.word_tokenize(s2)

['``', 'We', 'beat',
'some', 'pretty', 'good',
'teams', 'to', 'get',
'here', ',',
 "''", 'Slocum', 'said',
'.']
```

Part of Speech (POS)

Parts of speech are **word classes** or **lexical categories**

Nouns, verbs, adjectives, prepositions, etc.
First named by Dionysius **Thrax** of Alexandria (~100 BCE)

POS **annotation** is often called **tagging**, as in NLTK

POS tags **pertain to tokens**, not to vocabulary terms per se

The same word may function as a verb or a noun — "book a flight" "read a book"

Also, we often use verbs as nouns and nouns as verbs — "What's the ask?" "What's the spend on that?"

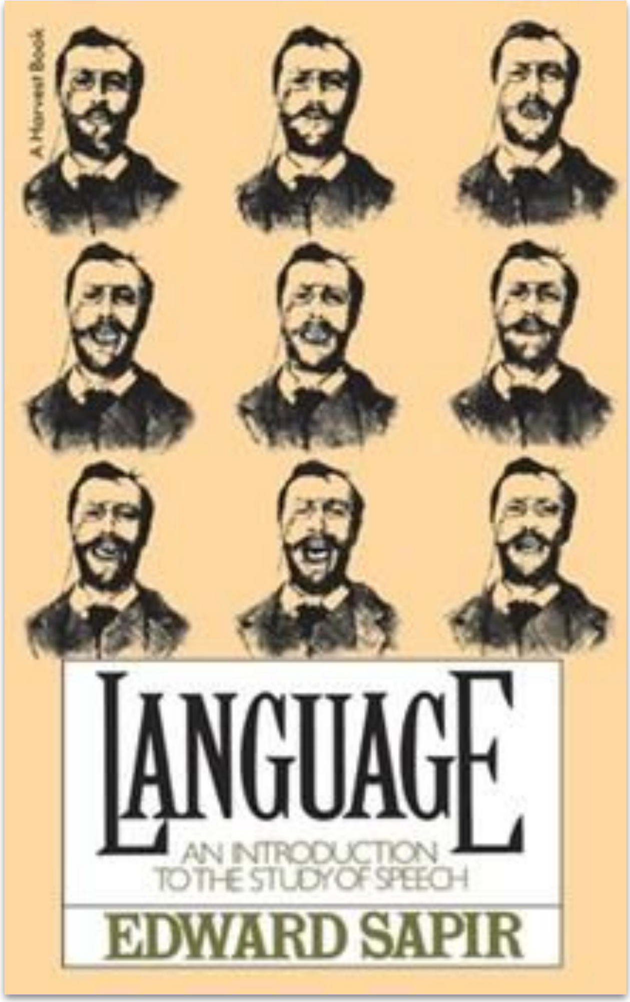
Thus, POS tagging is ***disambiguation*** task

POS Tagging

Process of **classifying** tokens by their part of speech

A **disambiguation** task

> Words are ambiguous
>
> **"All grammars leak."**
>
> — Edward Sapir
>
> i.e. word meanings and grammatical roles can shift and change over time




SUCH BIRTHDAY, MUCH AMAZE
SO WOW!
quickmeme.com

POS Ambiguity and Frequency

		WSJ		Brown	
Types:					
Unambiguous	(1 tag)	44,432	(**86%**)	45,799	(**85%**)
Ambiguous	(2+ tags)	7,025	(**14%**)	8,050	(**15%**)
Tokens:					
Unambiguous	(1 tag)	577,421	(**45%**)	384,349	(**33%**)
Ambiguous	(2+ tags)	711,780	(**55%**)	786,646	(**67%**)

Figure 8.2 Tag ambiguity for word types in Brown and WSJ, using Treebank-3 (45-tag) tagging. Punctuation were treated as words, and words were kept in their original case.

So, there are **fewer ambiguous terms**, but they are **more frequently used**

- **Proper nouns** (i.e. names) are not ambiguous

Thus, POS **ambiguity**, a form of polysemy, appears to be correlated with **frequency**

- We will **revisit this** later in the course when we discuss word embedding

POS Ambiguity and Frequency

Some of the most ambiguous frequent words are *that*, *back*, *down*, *put* and *set*; here are some examples of the 6 different parts-of-speech for the word *back*:

earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** about debt
I was twenty-one **back/RB** then

➤ Note that some linguists (e.g. Ruhl, 1989, *On Monosemy*), argue that **such words are actually monosemic** – they each have a single general meaning that is applied in different situations

Dictionary makers treat these usages as different meanings

POS Ambiguity

One way to disambiguate POS is to choose most frequently associated with a term

Most Frequent Class Baseline

We will generate this as a **feature of the VOCAB** table

POS Corpora – Tagged by POS

Brown

The **Brown University Standard Corpus of Present-Day American English**

500 samples of English-language text, one million words (1961)

WSJ

~25 million word parsed text from WSJ (1987-89)

Switchboard-1

~2,400 two-sided telephone conversations among 543 speakers with about 70 provided conversation topics (1992-93)

The Penn Treebank

A POS-annotated corpus with over **4.5 million words** of American English

- Based on 2,499 **stories** from the WSJ collection

Developed a **code** for POS tagging that is considered standard

- Tagset based on Brown corpus but pared down
- Strives to eliminate lexical redundancy

Should POS distributions vary
by **author**, speech **community,** or **language**?

Sure ... Some authors use
lots of adjectives, other don't

Some authors create lots of noun phrases

Compare Jane Austen to William Gibson
(author of *Neuromancer*)

Analytics vs Synthetic Languages

POS represented differently by languages

Synthetic languages have a high **morpheme-to-word ratio**

Grammatical work done **inflections**, **agglutinations**, etc.

Analytic languages have a low morpheme-to-word ratio

Work done with higher use of **auxiliary verbs** and **word order**

Languages vary in the use of **morphology** (word shape) and **syntax** (word order)

Morpho-syntactics

Penn Treebank Part-of-Speech Tags

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*’s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential ‘there’	*there*	PRP$	possess. pronoun	*your, one’s*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	*+,%, &*	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Figure 8.1 Penn Treebank part-of-speech tags (including punctuation).

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Alphabetical list of POS tags used in the Penn Treebank Project. To get info on each tag within NLTK, run this:

```
nltk.help.upenn_tagset()
```

POS Labeling Conventions

By convention

`<token>/<POS>`

e.g. **`The/DT gran/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.`**

We will generally treat POS as a feature in the TOKEN and VOCAB tables

Again, in VOCAB it will be most frequent

Use of POS

In ETA, POS is a **hyperparameter**, like OHCO

Just as some models work better with paragraphs or sentences

So do some models work better with nouns only, or with stopwords not removed

Examples:

Topic models work well with **paragraphs** and **nouns**

Word embeddings work will with **sentences** and **all words**

POS Tagging Methods

Hidden Markov Models (HMM)

- Sequence model – maps a sequence of tokens to a series of labels
- Similar to bigram language model

Maximum Entropy Markov Models (MEMMs)

As mentioned earlier, we will **not** be creating our own NLP taggers, such as a POS tagger

But it is helpful to **understand** how they are created

The **intuition**, generative **model**, **estimation** of parameters, and **application**

Hidden Markov Models (HMMs) for POS Prediction

Visible and Hidden Markov Models

The Ngram models we looked at last week are sometimes called **visible Markov models**

- The operate on a **single stream** of observed tokens

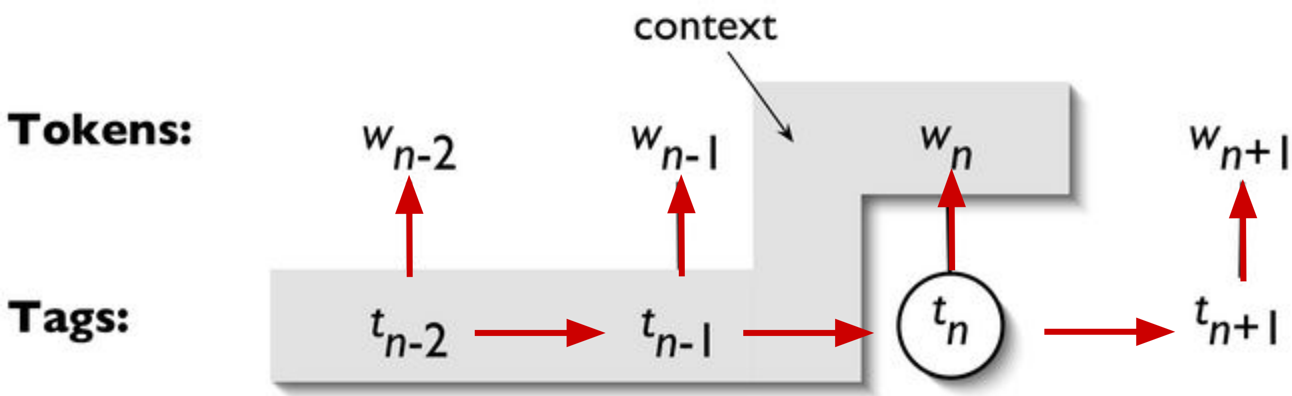
To predict annotations like part-of-speech, we use **hidden Markov models** (**HMM**s)

- These operate on **parallel streams**, where one is **visible** and the other is **latent**

Both are based on the same principle

- Reducing the **historical horizon** of an event, which **reduces the space of the chain rule** and simplifies the computation of conditional probabilities

Applied HMM: POS Tagging by N-Grams



An N-gram tagger's context is the **current word token** (w_n) together with the **preceding part-of-speech tags** (t_{n-1}, ... t_{n-N+1})

TRANSITION
from t_{n-1} to t_n

EMISSION
from t_n to w_n

					pos	term_str
book_id	chap_num	para_num	sent_num	token_num		
158	1	1	0	0	NNP	emma
				1	NNP	woodhouse
				2	NN	handsome
				3	NN	clever
				4	CC	and
				5	NN	rich
				6	IN	with
				7	DT	a
				8	JJ	comfortable
				9	NN	home
				10	CC	and
				11	JJ	happy
				12	NN	disposition
				13	VBD	seemed
				14	TO	to
				15	VB	unite
				16	DT	some

Axis of
EMISSION

Axis of
TRANSITION

As we will see, the **data** and its **structure** are already in place in the **TOKEN** table, so we can easily implement an HMM model :-)

HMM Formula and Estimation

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}}$$

For each row in TOKENS, find the highest probability POS tag

Probability of a tag given a preceding tag

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80$$

Probability of a word given a tag

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \quad P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31$$

(See Jurafsky and Martin, Chapter 8)

Viterbi

HMM is usually implemented with the Viterbi algorithm

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

```
function viterbi(O, S, Π, Tm, Em) : best_path      Tm: transition matrix   Em: emission matrix
    trellis ← matrix(length(S), length(O))          To hold probability of each state given each observation
    pointers ← matrix(length(S), length(O))         To hold backpointer to best prior state
    for s in range(length(S)):                      Determine each hidden state's probability at time 0…
        trellis[s, 0] ← Π[s] · Em[s, O[0]]
    for o in range(1, length(O)):                   …and after, tracking each state's most likely prior state, k
        for s in range(length(S)):
            k ← arg max(trellis[k, o − 1] · Tm[k, s] · Em[s, o] for k in range(length(S)))
            trellis[s, o] ← trellis[k, o − 1] · Tm[k, s] · Em[s, o]
            pointers[s, o] ← k
    best_path ← list()
    k ← arg max(trellis[k, length(O) − 1] for k in range(length(S)))     Find k of best final state
    for o in range(length(O) − 1, −1, −1):          Backtrack from last observation
        best_path.insert(0, S[k])                   Insert previous state on most likely path
        k ← pointers[k, o]                          Use backpointer to find best previous state
    return best_path
```

https://en.wikipedia.org/wiki/Viterbi_algorithm

A Note about Conditional Probability

$$P(b \mid a) = P(b, a) / P(a)$$

a : **antecedent**

acts as a WHERE statement the filters a **subset**

b : **consequent**

acts as a SELECT statement

$P(b|a) \rightarrow$ **`SELECT b ... WHERE a`**

This has two **advantages**:

We can think of probability in terms of data manipulation (SQL)

We avoid temporalizing the relationship – it's really structural

Conditional Probability in Pandas

$P(w1|w0)$ where dataframe **`m2`** has

index **`w0, w1`** and column **`n`**

With **`df.groupby()`**

```
m2.n / m2.groupby('w0').n.sum()
```

Or, if dataframe **`m1`** has index **`w0`** and column **`n`**

```
m2.n / m1.n
```

Conditional Probability in Pandas with `unstack()`

w1		1	11th	12th	13th	17	18th	19	2	26th	...	younger	youngest	youngster	your	yours	yourself	yourselves
w0																		
	22	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
11th	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0
12th	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0
13th	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
yourself	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
yourselves	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
youth	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
youthful	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
zeal	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

Σ

Σ

$P(w0 \mid w1)$ **`axis = 0`**

$P(w1 \mid w0)$ **`axis = 1`**

The conditioning event defines the axis of normalization (summing)