# Natural Language Processing for Law and Social Science

2. Tokenization

## Homework & TA Sessions

- First homework is due Thursday by midnight.
  - Submit IPYNB file on EduFlow (reachable from moodle).
  - Completion grade – full credit for trying every question and submitting on time (checked programmatically and by random audit)
  - Have to submit an assignment (even if late) to see example solution.

## Homework & TA Sessions

- First homework is due Thursday by midnight.
  - Submit IPYNB file on EduFlow (reachable from moodle).
  - Completion grade – full credit for trying every question and submitting on time (checked programmatically and by random audit)
  - Have to submit an assignment (even if late) to see example solution.

- TA Office Hours: Thursdays, 10am-11am
- TA Sessions: Fridays, 10am-11am

## Homework & TA Sessions

- First homework is due Thursday by midnight.
  - Submit IPYNB file on EduFlow (reachable from moodle).
  - Completion grade – full credit for trying every question and submitting on time (checked programmatically and by random audit)
  - Have to submit an assignment (even if late) to see example solution.

- TA Office Hours: Thursdays, 10am-11am
- TA Sessions: Fridays, 10am-11am

**Any other logistical questions?**

## In-Class Presentation: Michalopoulos and Xue, “Folkore” (2021)

## Tokenization: Overview

- Input:
  - A set of documents (e.g. text files), $D$.
- Output 1:
  - A sequence, $W$, containing a list of tokens – words or word pieces for use in natural language processing
- Output 2:
  - A document-term matrix, $X$, containing statistics about word/phrase frequencies in those documents.

## Three Approaches to Tokenization

1. convert documents to count vectors, e.g. over pre-processed n-grams.
   - “bag of words” or “bag of terms” representation
   - used with topic models and classical ML
   - should be **informative/predictive** in the learning task, computationally **tractable**, and preferably somewhat **interpretable**.

## Three Approaches to Tokenization

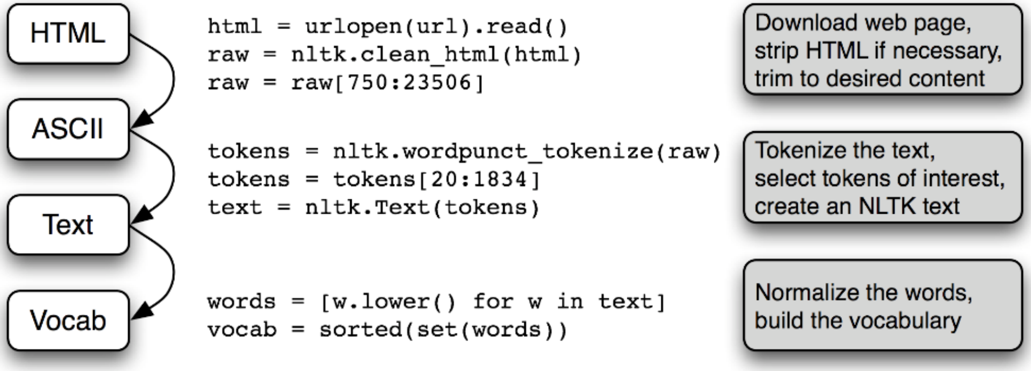
1. convert documents to count vectors, e.g. over pre-processed n-grams.
   - "bag of words" or "bag of terms" representation
   - used with topic models and classical ML
   - should be **informative/predictive** in the learning task, computationally **tractable**, and preferably somewhat **interpretable**.

2. segment documents into word pieces using byte pair encoding
   - maintain as much info from the original document as possible
   - for inputs to sequence models, i.e. transformers.

## Three Approaches to Tokenization

1. convert documents to count vectors, e.g. over pre-processed n-grams.
   - "bag of words" or "bag of terms" representation
   - used with topic models and classical ML
   - should be **informative/predictive** in the learning task, computationally **tractable**, and preferably somewhat **interpretable**.
2. segment documents into word pieces using byte pair encoding
   - maintain as much info from the original document as possible
   - for inputs to sequence models, i.e. transformers.
3. enrich document with linguistics/grammar information
   - add more information to unprocessed doc based on sentence boundaries, parts of speech, syntax, etc
   - needed for specific tasks – eg relation extraction

## A Standard Tokenization Pipeline


HTML
ASCII
Text
Vocab
Download web page, strip HTML if necessary, trim to desired content
Tokenize the text, select tokens of interest, create an NLTK text
Normalize the words, build the vocabulary

```
html = urlopen(url).read()
raw = nltk.clean_html(html)
raw = raw[750:23506]

tokens = nltk.wordpunct_tokenize(raw)
tokens = tokens[20:1834]
text = nltk.Text(tokens)

words = [w.lower() for w in text]
vocab = sorted(set(words))
```

Source: NLTK Book, Chapter 3.

## Pre-processing

- For many projects, the first question is: what data to throw out?
  - Uninformative data add noise and reduce statistical precision.
  - They are also computationally costly.
- Pre-processing choices can affect down-stream results, especially in unsupervised learning tasks (Denny and Spirling 2017).
  - in particular: some features are more interpretable, e.g. (“discretion”, “have”, “judge”) vs (“the judge has discretion”).

## Normalizing Text

1. Capitalization
   - usually the capitalized/non-capitalized version of a word are equivalent $\rightarrow$ capitalization not informative.

## Normalizing Text

1. Capitalization
   - usually the capitalized/non-capitalized version of a word are equivalent $\rightarrow$ capitalization not informative.
   - On the other hand: what about “the first amendment” versus “the First Amendment”?

## Normalizing Text

1. Capitalization
   - usually the capitalized/non-capitalized version of a word are equivalent $\rightarrow$ capitalization not informative.
   - On the other hand: what about “the first amendment” versus “the First Amendment”?
2. Punctuation
   - the number of periods or commas in a document is usually not that useful
   - so in a bag of terms approach, punctuation can be dropped.

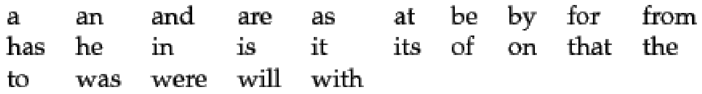
## Normalizing Text

1. Capitalization
   - usually the capitalized/non-capitalized version of a word are equivalent $\rightarrow$ capitalization not informative.
   - On the other hand: what about “the first amendment” versus “the First Amendment”?
2. Punctuation
   - the number of periods or commas in a document is usually not that useful
   - so in a bag of terms approach, punctuation can be dropped.
   - but what about “Let’s eat, Grandpa”, versus “Let’s eat Grandpa”?

## Normalizing Text

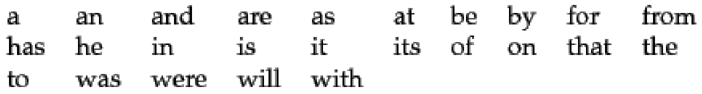
1. Capitalization
   - usually the capitalized/non-capitalized version of a word are equivalent $\rightarrow$ capitalization not informative.
   - On the other hand: what about “the first amendment” versus “the First Amendment”?
2. Punctuation
   - the number of periods or commas in a document is usually not that useful
   - so in a bag of terms approach, punctuation can be dropped.
   - but what about “Let’s eat, Grandpa”, versus “Let’s eat Grandpa”?
3. Numbers
   - individual numbers are usually too specific to keep in the vocabulary
   - But how often numbers are mentioned might be important; can replace with a special character, e.g. #.

## Stopwords



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | an | and | are | as | at | be | by | for | from |
| has | he | in | is | it | its | of | on | that | the |
| to | was | were | will | with | | | | | |

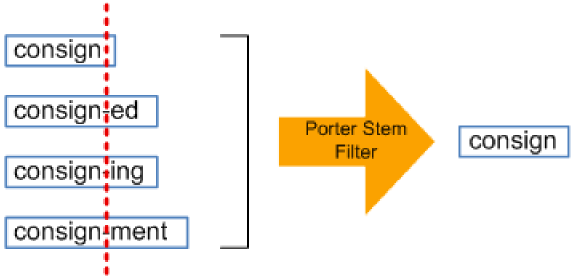
## Stopwords



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | an | and | are | as | at | be | by | for | from |
| has | he | in | is | it | its | of | on | that | the |
| to | was | were | will | with | | | | | |

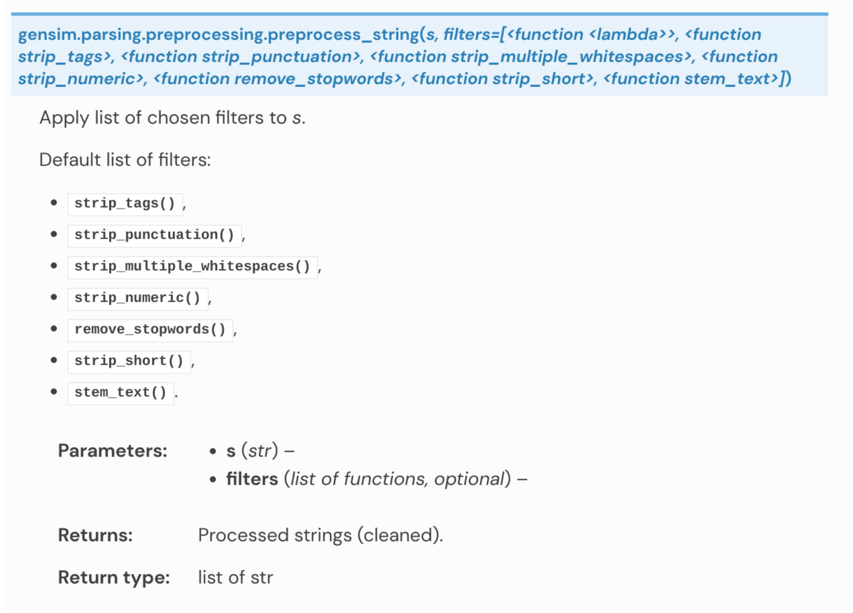
- What about “not guilty”?
- Legal terms often contain stopwords:
  - “beyond a reasonable doubt”
  - “with all deliberate speed”
- can drop stopwords by themselves, but keep them when part of phrases.

## Stemming/lemmatizing



- Effective dimension reduction with little loss of information.
- Lemmatizer produces real words, but N-grams won't make grammatical sense
  - e.g., "judges have been ruling" would become "judge have be rule"

## Pre-processing with `gensim`



**gensim.parsing.preprocessing.preprocess_string**(*s, filters=[<function <lambda>>, <function strip_tags>, <function strip_punctuation>, <function strip_multiple_whitespaces>, <function strip_numeric>, <function remove_stopwords>, <function strip_short>, <function stem_text>]*)

Apply list of chosen filters to *s*.

Default list of filters:

- `strip_tags()`,
- `strip_punctuation()`,
- `strip_multiple_whitespaces()`,
- `strip_numeric()`,
- `remove_stopwords()`,
- `strip_short()`,
- `stem_text()`.

**Parameters:**
- **s** (*str*) –
- **filters** (*list of functions, optional*) –

**Returns:** Processed strings (cleaned).

**Return type:** list of str

## Bag-of-words representation

Say we want to convert a corpus $D$ to a matrix $X$:

- In the "bag-of-words" representation, a row of $X$ is just the frequency distribution over words in the document corresponding to that row.

Counts and frequencies:

- **Document counts**: number of documents where a word appears.
- **Term counts**: number of total appearances of a word in corpus.
- **Term frequency**:

$$\text{Term Frequency of } w \text{ in document } k = \frac{\text{Count of } w \text{ in document } k}{\text{Total tokens in document } k}$$

## Building a vocabulary

- What are the columns of the document-term matrix $X$?
  - Assign numerical indices to words to increase speed and reduce disk usage.

- Pick a number:, e.g. 100,000 most frequent words.

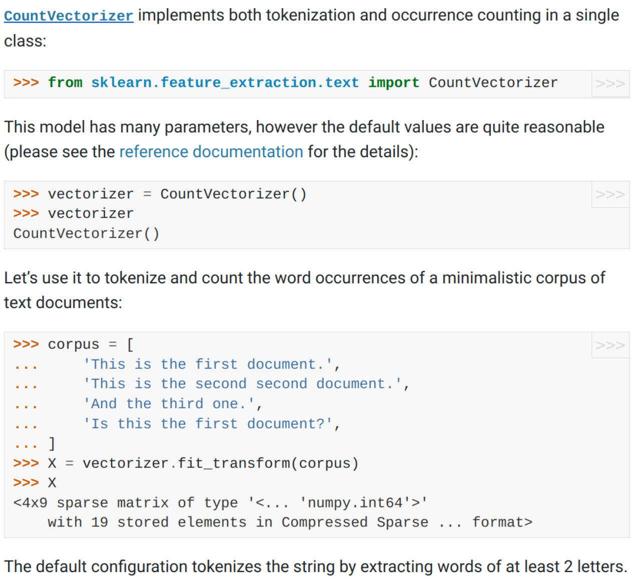
## Building a vocabulary

- What are the columns of the document-term matrix $X$?
  - Assign numerical indices to words to increase speed and reduce disk usage.

- Pick a number:, e.g. 100,000 most frequent words.
- Frequency threshold:
  - Compute document frequencies for all words
  - Inspect low-frequency words and determine a minimum document threshold.
    - e.g., 10 documents, or .25% of documents.

# scikit-learn's CountVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html



`CountVectorizer` implements both tokenization and occurrence counting in a single class:

```
>>> from sklearn.feature_extraction.text import CountVectorizer
```

This model has many parameters, however the default values are quite reasonable (please see the reference documentation for the details):

```
>>> vectorizer = CountVectorizer()
>>> vectorizer
CountVectorizer()
```

Let's use it to tokenize and count the word occurrences of a minimalistic corpus of text documents:

```
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third one.',
...     'Is this the first document?',
... ]
>>> X = vectorizer.fit_transform(corpus)
>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

The default configuration tokenizes the string by extracting words of at least 2 letters.

- `corpus` is a sequence of strings, e.g. pandas data-frame columns.
- pre-processing options: strip accents, lowercase, drop stopwords,
- vocab options: min/max frequency, vocab size
- n-grams: can produce phrases up to length n (words or characters).

## What about out-of-vocab words?

## What about out-of-vocab words?

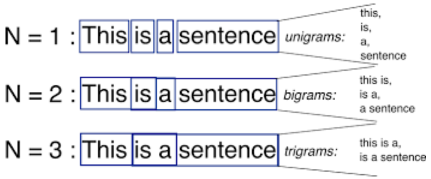
- in bag-of-words model:
  - drop them
  - replace with “unknown” token (<unk>)
  - replace with part-of-speech tag
  - replace with in-vocab hypernym (from WordNet)
  - others?

## What about out-of-vocab words?

- in bag-of-words model:
  - drop them
  - replace with “unknown” token (<unk>)
  - replace with part-of-speech tag
  - replace with in-vocab hypernym (from WordNet)
  - others?
- alternative approaches that don’t have this problem (more below):
  - hashing vectorizer
  - byte pair encoding

## N-grams are phrases, sequences of words up to length $N$

- e.g. bigrams, trigrams, quadgrams, etc.



- Baseline for text classification of long documents (Google Developers Guide):
  - $X$ = counts over bigrams, with vocab size = 20,000

`https://developers.google.com/machine-learning/guides/text-classification/step-3`

## Feature selection

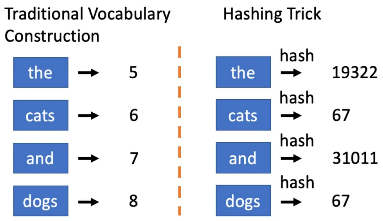
- N-grams quickly blow up the feature space.
  - filtering on frequency is easiest but not optimal – can filter on usefulness for a task instead.
- Text normalization is important (capitalization, punctuation, stopwords, stemming)

- For supervised learning tasks:
  - Use supervised feature selection to select predictive features (more on week 4)
- What about unsupervised learning (e.g. topic models)?
  - can use parts of speech / co-location statistics (week 3)

## Feature selection

- N-grams quickly blow up the feature space.
  - filtering on frequency is easiest but not optimal – can filter on usefulness for a task instead.
- Text normalization is important (capitalization, punctuation, stopwords, stemming)
- For supervised learning tasks:
  - Use supervised feature selection to select predictive features (more on week 4)
- What about unsupervised learning (e.g. topic models)?
  - can use parts of speech / co-location statistics (week 3)
- In week 3, we explore more general problem of dimensionality reduction.

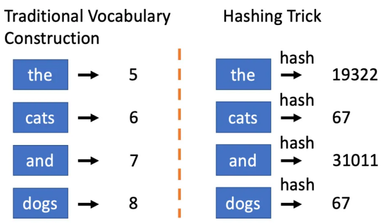
## Hashing Vectorizer



- Rather than make a one-to-one lookup for each n-gram, put n-grams through a hashing function that takes an arbitrary string and deterministically outputs an integer in some range (e.g. 1 to 10,000).

```
>>> from sklearn.feature_extraction.text import HashingVectorizer
>>> hv = HashingVectorizer(n_features=10)
>>> hv.transform(corpus)
<4x10 sparse matrix of type '<... 'numpy.float64'>'
    with 16 stored elements in Compressed Sparse ... format>
```

## Hashing Vectorizer



- Rather than make a one-to-one lookup for each n-gram, put n-grams through a hashing function that takes an arbitrary string and deterministically outputs an integer in some range (e.g. 1 to 10,000).

```
>>> from sklearn.feature_extraction.text import HashingVectorizer
>>> hv = HashingVectorizer(n_features=10)
>>> hv.transform(corpus)
<4x10 sparse matrix of type '<... 'numpy.float64'>'
    with 16 stored elements in Compressed Sparse ... format>
```

Pros:

- can have arbitrarilly small feature space
- handles out-of-vocabulary words – any word or n-gram gets assigned to an arbitrary integer based on the hash function.

Cons:

- harder to interpret features, at least not directly (eli5 implementation keeps track of the mapping)-
- collisions – n-grams will randomly be paired with each other in the feature map – in supervised learning, usually innocuous

## In-Class Presentation: Gentzkow and Shapiro (2010) “What Drives Media Slant?”

## Limitations of word tokenization

- modern language models are designed/intended to capture all meaning in texts.
  - with word tokenization, would need a massive vocabulary to capture all cases.
  - not impossible but computationally costly.
  - model might encounter new words in the test data.

## Limitations of word tokenization

- modern language models are designed/intended to capture all meaning in texts.
  - with word tokenization, would need a massive vocabulary to capture all cases.
  - not impossible but computationally costly.
  - model might encounter new words in the test data.
- treats different word forms as separate words (e.g. “tax”, “taxes”, “taxed”)
  - or, with stemming, treats them as identical

## Character tokenization

- alternative – tokenize characters rather than words:
    - “hello world” $\rightarrow$ $\{$h,e,l,l,o, ,w,o,r,l,d$\}$
    - by construction, no unkown words.

## Character tokenization

- alternative – tokenize characters rather than words:
  - “hello world” $\rightarrow$ $\{$h,e,l,l,o, ,w,o,r,l,d$\}$
  - by construction, no unkown words.
- this actually works fine, and is used in some recent language models.

## Character tokenization

- alternative – tokenize characters rather than words:
  - "hello world" $\rightarrow$ $\{$h,e,l,l,o, ,w,o,r,l,d$\}$
  - by construction, no unkown words.
- this actually works fine, and is used in some recent language models.
  - but not efficient: some single characters (e.g. "x") are much less frequent than some character subsequences (e.g. "tion") or even whole words (e.g. "the")

## Subword Tokenization for Sequence Models

Most modern language models (e.g. BERT, GPT) use subword tokenization:

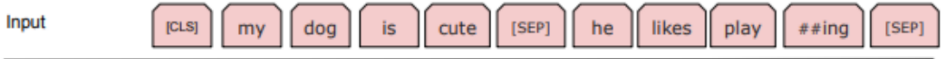
- construct character-level n-grams using byte pair encoding (frequent character sequences treated as a token)
- whitespace treated the same as letters
- all letters to lowercase, but add a special character for the next letter being capitalized.

## Subword Tokenization for Sequence Models

Most modern language models (e.g. BERT, GPT) use subword tokenization:

- construct character-level n-grams using byte pair encoding (frequent character sequences treated as a token)
- whitespace treated the same as letters
- all letters to lowercase, but add a special character for the next letter being capitalized.

e.g., BERT's SentencePiece tokenizer:



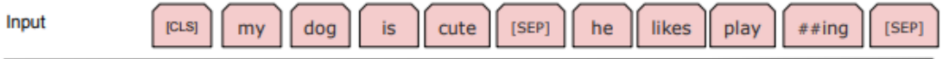
- character-level byte-pair encoder, learns character n-grams to breaks words like "playing" into "play" and "##ing".
- have to fix a vocabulary size: e.g. BERT uses 30K.
- see notebook

## Subword Tokenization for Sequence Models

Most modern language models (e.g. BERT, GPT) use subword tokenization:

- construct character-level n-grams using byte pair encoding (frequent character sequences treated as a token)
- whitespace treated the same as letters
- all letters to lowercase, but add a special character for the next letter being capitalized.

e.g., BERT's SentencePiece tokenizer:



- character-level byte-pair encoder, learns character n-grams to breaks words like "playing" into "play" and "##ing".
- have to fix a vocabulary size: e.g. BERT uses 30K.
- see notebook

GPT-3 tokenizer: `https://platform.openai.com/tokenizer`

## Limitations of word/subword tokenization

- The tokenizers we have looked at so far are based only on data in the corpus.
  - bag-of-words tokenizers break down text into counts over the most relevant features.
  - subword tokenizers try to preserve the text as is.

## Limitations of word/subword tokenization

- The tokenizers we have looked at so far are based only on data in the corpus.
  - bag-of-words tokenizers break down text into counts over the most relevant features.
  - subword tokenizers try to preserve the text as is.
- These tokenizers leave out a lot of information that we have from sophisticated and powerful conceptual models of language – that is, linguistics.

## Segmenting paragraphs/sentences

- Many tasks should be done on sentences, rather than corpora as a whole.
  - spaCy is a good (but not perfect) job of splitting sentences, while accounting for periods on abbreviations, etc.
- There isn't a grammar-based paragraph tokenizer.
  - most corpora have new paragraphs annotated.
  - or use line breaks.

## Parts of speech tags

- Parts of speech (POS) tags provide useful word categories corresponding to their functions in sentences:
  - Eight main parts of speech: verb (VB), noun (NN), pronoun (PR), adjective (JJ), adverb (RB), determinant (DT), preposition (IN), conjunction (CC).
  - The Penn TreeBank POS tag set (used in many applications) has 36 tags: `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

## Parts of speech tags

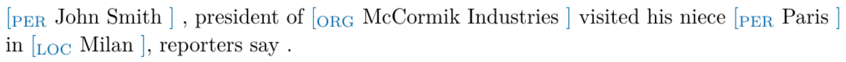
- Parts of speech (POS) tags provide useful word categories corresponding to their functions in sentences:
    - Eight main parts of speech: verb (VB), noun (NN), pronoun (PR), adjective (JJ), adverb (RB), determinant (DT), preposition (IN), conjunction (CC).
    - The Penn TreeBank POS tag set (used in many applications) has 36 tags: `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`
- Parts of speech vary in their informativeness for various functions:
    - For categorizing topics, nouns are usually most important
    - For sentiment, adjectives are usually most important.
- In particular, noun phrases are often informative features – spaCy can do fast noun phrase chunking.

## Parts of speech tags

- Parts of speech (POS) tags provide useful word categories corresponding to their functions in sentences:
  - Eight main parts of speech: verb (VB), noun (NN), pronoun (PR), adjective (JJ), adverb (RB), determinant (DT), preposition (IN), conjunction (CC).
  - The Penn TreeBank POS tag set (used in many applications) has 36 tags: `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`
- Parts of speech vary in their informativeness for various functions:
  - For categorizing topics, nouns are usually most important
  - For sentiment, adjectives are usually most important.
- In particular, noun phrases are often informative features – spaCy can do fast noun phrase chunking.
- Can count POS tags as features – e.g., using more adjectives, or using more passive verbs.
  - provides style features, e.g. for authorship detection.

## Named Entity Recognition

- Named entities such as “ETH Zurich” and “Marie Curie” are a special set of annotations, tagged by named entity recognizers (NER).
- usually identified by proper nouns; most pre-trained NER systems, like spACy, also give an entity category:



[PER John Smith ] , president of [ORG McCormik Industries ] visited his niece [PER Paris ] in [LOC Milan ], reporters say .

## Application: POS tags Predict Loan Repayment

Netzer, Lemaire, and Herzenstein (2019), “When Words Sweat”

## Application: POS tags Predict Loan Repayment

Netzer, Lemaire, and Herzenstein (2019), "When Words Sweat"

- Imagine you consider lending $2,000 to one of two borrowers on a crowdfunding website. The borrowers are identical in terms of demographic and financial characteristics. However, the text they provided when applying for a loan differs:
  - Borrower #1: "*I am a hard working person, married for 25 years, and have two wonderful boys. Please let me explain why I need help. I would use the $2,000 loan to fix our roof. Thank you, god bless you, and I promise to pay you back.*"
  - Borrower #2: "*While the past year in our new place has been more than great, the roof is now leaking and I need to borrow $2,000 to cover the cost of the repair. I pay all bills (e.g., car loans, cable, utilities) on time.*"
- Which borrower is more likely to default?

## Application: POS tags Predict Loan Repayment

Netzer, Lemaire, and Herzenstein (2019), "When Words Sweat"

- Imagine you consider lending \$2,000 to one of two borrowers on a crowdfunding website. The borrowers are identical in terms of demographic and financial characteristics. However, the text they provided when applying for a loan differs:
  - Borrower #1: "*I am a hard working person, married for 25 years, and have two wonderful boys. Please let me explain why I need help. I would use the \$2,000 loan to fix our roof. Thank you, god bless you, and I promise to pay you back.*"
  - Borrower #2: "*While the past year in our new place has been more than great, the roof is now leaking and I need to borrow \$2,000 to cover the cost of the repair. I pay all bills (e.g., car loans, cable, utilities) on time.*"
- Which borrower is more likely to default?

**"Loan requests written by defaulting borrowers are more likely to include words (or themes) related to the borrower's family, financial and general hardship, mentions of god, and the near future, as well as pleading lenders for help, and using verbs in present and future tenses."**

## Loan Application Words Predicting Repayment (Netzer, Lemaire, and Herzenstein 2019)

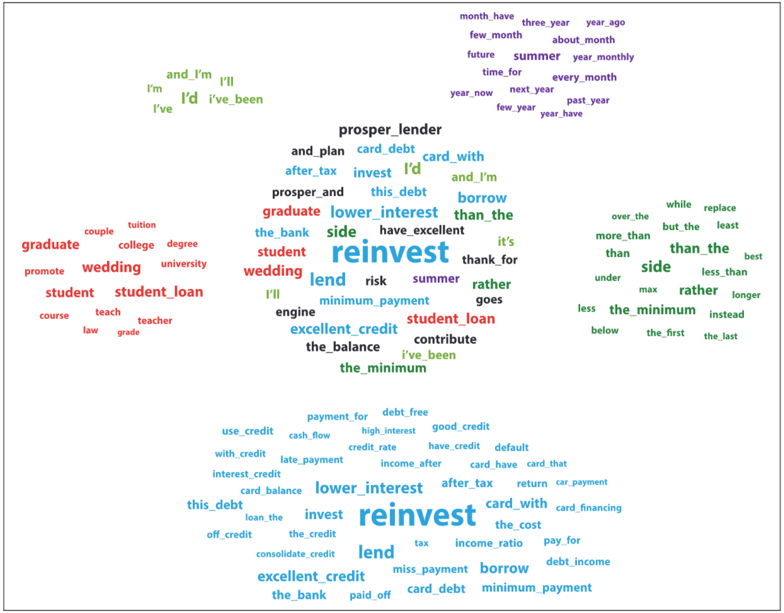


**Figure 2.** Words indicative of loan repayment.

*Notes:* The most common words appear in the middle cloud (cutoff = 1:1.5) and are then organized by themes. Starting on the right and moving clockwise: relative words, financial literacy words, words related to a brighter financial future, "I" words, and time-related words.

# Loan Application Words Predicting Default (Netzer, Lemaire, and Herzenstein 2019)

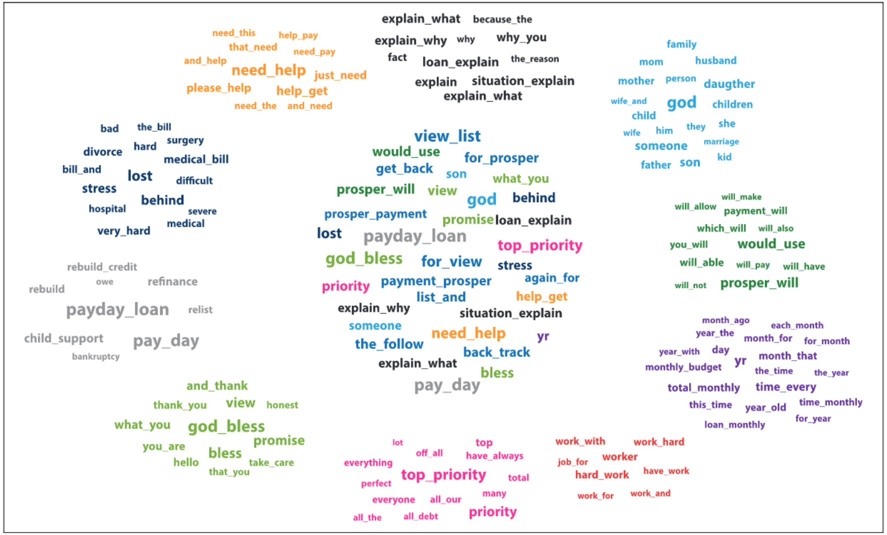

explain_what because_the
explain_why why why_you
fact loan_explain the_reason
explain situation_explain
explain_what
need_this help_pay that_need need_pay and_help need_help just_need please_help help_get need_the and_need
family mom husband mother person daugther wife_and god children child wife him they she someone marriage father son kid
bad the_bill surgery divorce hard medical_bill bill_and lost difficult stress behind severe hospital medical very_hard
view_list would_use for_prosper get_back son what_you prosper_will view god behind prosper_payment promise loan_explain lost payday_loan top_priority god_bless for_view stress priority payment_prosper again_for list_and help_get explain_why situation_explain someone need_help yr the_follow back_track explain_what bless pay_day
will_make will_allow payment_will which_will will_also you_will would_use will_able will_pay will_have will_not prosper_will
rebuild_credit owe refinance rebuild payday_loan relist child_support pay_day bankruptcy
month_ago each_month year_the month_for for_month year_with day month_that monthly_budget yr the_time the_year total_monthly time_every this_time year_old time_monthly loan_monthly for_year
and_thank thank_you view honest what_you god_bless promise you_are bless hello take_care that_you
lot top off_all have_always everything top_priority total perfect many everyone all_our all_the all_debt priority
work_with work_hard job_for worker hard_work have_work work_for work_and

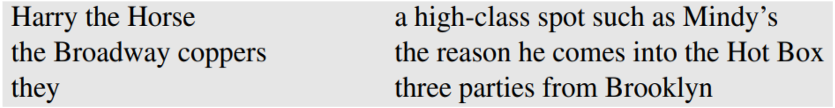
**Figure 3.** Words indicative of loan default.

*Notes*: The most common words appear in the middle cloud (cutoff = 1:1.5) and are then organized by themes. Starting on the top and moving clockwise: words related to explanations, external influence words and others, future-tense words, time-related words, work-related words, extremity words, words appealing to lenders, words relating to financial hardship, words relating to general hardship, and desperation/plea words.

## Using Grammar: Constituency

## Using Grammar: Constituency

- The idea of constituency is that groups of words behave as singular functional units in a sentence.
- Some example noun phrases:

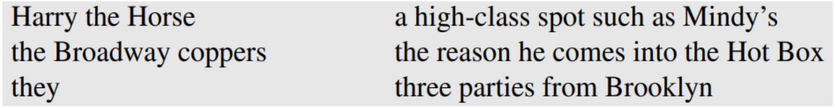


| | |
|---|---|
| Harry the Horse | a high-class spot such as Mindy's |
| the Broadway coppers | the reason he comes into the Hot Box |
| they | three parties from Brooklyn |

  - these phrases consist of many POS's but function as nouns

## Using Grammar: Constituency

- The idea of constituency is that groups of words behave as singular functional units in a sentence.
- Some example noun phrases:



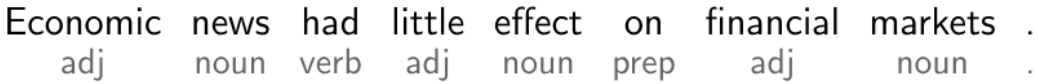
| | |
|---|---|
| Harry the Horse | a high-class spot such as Mindy's |
| the Broadway coppers | the reason he comes into the Hot Box |
| they | three parties from Brooklyn |

  - these phrases consist of many POS's but function as nouns
- In English, constituents can be moved around in a sentence (e.g. these prepositional phrases):
  - John talked [to the students] [about linguistics].
  - John talked [about linguistics] [to the students] .

## Using Grammar: Syntactic Dependencies

- The basic idea:
  - **Syntactic structure** consists of **words**, linked by binary directed relations called **dependencies**.
  - Dependencies identify the grammatical relations between words.

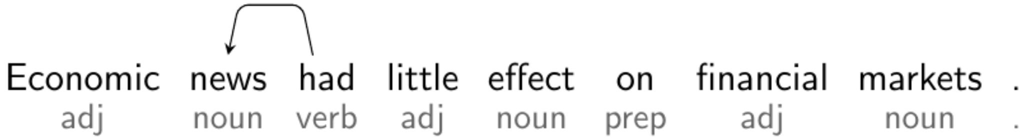
## Dependencies: Binary Directed Relations Between Words (Head and Dependent)



| Economic | news | had | little | effect | on | financial | markets | . |
|---|---|---|---|---|---|---|---|---|
| adj | noun | verb | adj | noun | prep | adj | noun | . |

- dependency trees are mostly determined by the ordering of POS tags.

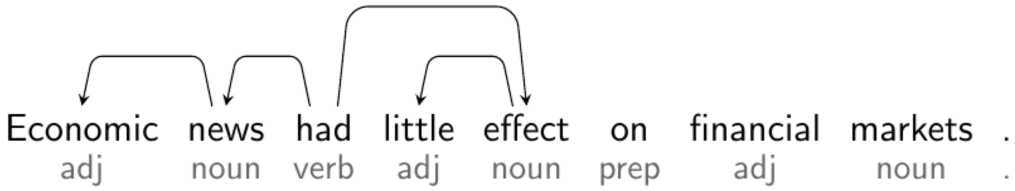
## Dependencies: Binary Directed Relations Between Words (Head and Dependent)



Economic news had little effect on financial markets .
adj noun verb adj noun prep adj noun .

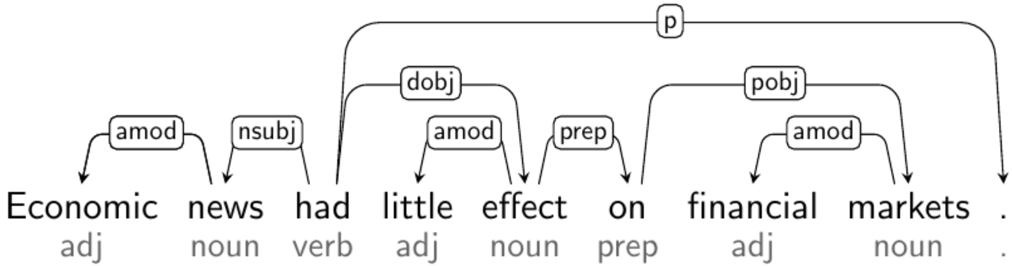
- the "root" of a sentence is the main verb (for compound sentences, the first verb).

## Dependencies: Binary Directed Relations Between Words (Head and Dependent)



- directed arcs indicate dependencies: a one-way link from a “head” token to a “dependent” token.
- A word can be “head” multiple times, but “dependent” only once.

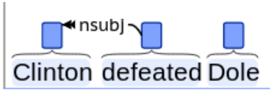
## Dependencies: Binary Directed Relations Between Words (Head and Dependent)



- arc labels indicate functional relations, e.g.:
  - nsubj: verb → subject doing the verb
  - dobj: verb → object targeted by the verb
  - amod: noun → attribute of the noun
- spaCy dependency visualizer: `https://explosion.ai/demos/displacy`

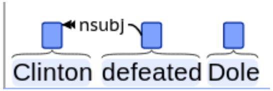
## Who does What to Whom: Subjects

- **nsubj: nominal subject**
  - points from the active verb to the agent subject.

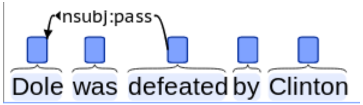


## Who does What to Whom: Subjects

- **nsubj: nominal subject**
  - points from the active verb to the agent subject.



- **nsubjpass: passive nominal subject**
  - points from a passive veb to the patient subject



## Who does What to Whom: Objects

**dobj: direct object**

- points from an active verb to the the (accusative) object noun phrase.

“She **gave** me a **raise**”

$$\textit{gave} \overset{\textit{dobj}}{\rightarrow} \textit{raise}$$

## Who does What to Whom: Objects

**dobj: direct object**

- points from an active verb to the the (accusative) object noun phrase.

“She **gave** me a **raise**”

$$\textit{gave} \overset{\textit{dobj}}{\rightarrow} \textit{raise}$$

**dative: dative or indirect object**

- points from an active verb to the the (dative) object noun phrase.

“She **gave me** a raise”

$$\textit{gave} \overset{\textit{dative}}{\rightarrow} \textit{me}$$

## Who does What to Whom: Objects

**dobj: direct object**

- points from an active verb to the the (accusative) object noun phrase.

“She **gave** me a **raise**”

$$\textit{gave} \overset{\textit{dobj}}{\rightarrow} \textit{raise}$$

**dative: dative or indirect object**

- points from an active verb to the the (dative) object noun phrase.

“She **gave me** a raise”

$$\textit{gave} \overset{\textit{dative}}{\rightarrow} \textit{me}$$

**pobj: object of a preposition**

- noun phrase following a preposition

“I sat **on** the **chair**”

$$\textit{on} \overset{\textit{pobj}}{\rightarrow} \textit{chair}$$

## What Attributes do Entities Have?

**acomp: adjectival complement**

- points from verb to adjectival phrase functioning as object

  "Bill **is honest** ": acomp(is $\rightarrow$ honest)

## What Attributes do Entities Have?

**acomp: adjectival complement**

- points from verb to adjectival phrase functioning as object

  "Bill **is honest** ": acomp(is $\rightarrow$ honest)

**attr: attribute**

- points from copula verb to an attribute noun phrase.

  "Bill **is** a **saint**": attr(is $\rightarrow$ saint)

## What Attributes do Entities Have?

**acomp: adjectival complement**

- points from verb to adjectival phrase functioning as object

  “Bill **is honest** ”: acomp(is $\rightarrow$ honest)

**attr: attribute**

- points from copula verb to an attribute noun phrase.

  “Bill **is** a **saint**”: attr(is $\rightarrow$ saint)

**amod: adjectival modifier**

- points from a noun to an adjective modifying it

  “Sam eats **red meat**”: amod(meat $\rightarrow$ red)

## Verb phrases

- **aux: auxiliary**
  - points from a main verb to a helping verb, including modals.

“Reagan **has died**”: aux(died $\rightarrow$ has)

“He **should leave**”: aux(leave $\rightarrow$ should)

## Verb phrases

- **aux: auxiliary**
  - points from a main verb to a helping verb, including modals.

“Reagan **has died**”: aux(died $\rightarrow$ has)

“He **should leave**”: aux(leave $\rightarrow$ should)

- **auxpass: passive auxiliary**
  - points from a main verb to a helping verb indicative passive voice.

“Laws have **been broken**”: auxpass(broken $\rightarrow$ been)

## Verb phrases

- **aux: auxiliary**
  - points from a main verb to a helping verb, including modals.

    “Reagan **has died**”: aux(died → has)

    “He **should leave**”: aux(leave → should)
- **auxpass: passive auxiliary**
  - points from a main verb to a helping verb indicative passive voice.

    “Laws have **been broken**”: auxpass(broken → been)
- **neg: negation modifier**
  - points from a verb to a negation indicator
  - “Bill **is not** a scientist”: neg(is → not)

## Verb phrases

- **aux: auxiliary**
  - points from a main verb to a helping verb, including modals.

    “Reagan **has died**”: aux(died $\rightarrow$ has)

    “He **should leave**”: aux(leave $\rightarrow$ should)
- **auxpass: passive auxiliary**
  - points from a main verb to a helping verb indicative passive voice.

    “Laws have **been broken**”: auxpass(broken $\rightarrow$ been)
- **neg: negation modifier**
  - points from a verb to a negation indicator
  - “Bill **is not** a scientist”: neg(is $\rightarrow$ not)
- **prt: phrasal verb particle**
  - points from a verb to its particle, linking phrasal verbs.

    “They **shut down** the station”: prt(shut $\rightarrow$ down)

## Verb phrases

- **aux: auxiliary**
  - points from a main verb to a helping verb, including modals.

    "Reagan **has died**": aux(died $\rightarrow$ has)

    "He **should leave**": aux(leave $\rightarrow$ should)
- **auxpass: passive auxiliary**
  - points from a main verb to a helping verb indicative passive voice.

    "Laws have **been broken**": auxpass(broken $\rightarrow$ been)
- **neg: negation modifier**
  - points from a verb to a negation indicator
  - "Bill **is not** a scientist": neg(is $\rightarrow$ not)
- **prt: phrasal verb particle**
  - points from a verb to its particle, linking phrasal verbs.

    "They **shut down** the station": prt(shut $\rightarrow$ down)
- and more...

## In-Class Presentation: Ash, Jacobs, MacLeod, Naidu, and Stammbach (2020) "Unsupervised extraction of rights and duties from collective bargaining agreements"

## Course Project Logistics

`https://bit.ly/NLP-proj`

- If you are signed up for the credits, the focus of your work in this course should be on the project.
  - Can be done individually or in small groups (up to 4 students).
  - Do an original analysis using methods learned in the course, and write a paper about it.

## Previous Year's Projects (1)

- One of the groups began building a legal research application for Swiss lawyers:
  - see `https://deepjudge.ai/`
  - feature-rich legal search engine, won some VC funding and now part of ETH AI Center
- Another group partnered with a local company to build out environmental-regulation analytics
  - won an Innosuisse grant.

## Previous Year's Projects (2)

**Selection of published projects:**

1. "Legal language modeling with transformers" (Lazar Peric, Stefan Mijic, Dominik Stammbach, Elliott Ash), *Proceedings of ASAIL* (2020).
2. "Entropy in Legal Language" (Roland Friedrich, Mauro Luzzatto, Elliott Ash), *NLLP @ KDD* (2020).
3. "Towards Automated Anamnesis Summarization: BERT-based Models for Symptom Extraction" (Anton Schäfer, Nils Blach, Oliver Rausch, Maximilian Warm, Nils Krüger), *Machine Learning for Health at NeurIPS* (2020).
4. "Kwame: A Bilingual AI Teaching Assistant for Online SuaCode Courses" (George Boateng), *International Conference on AI in Education* (2021)
5. "MemSum: Extractive Summarization of Long Documents using Multi-Step Episodic Markov Decision Processes" (Nianlong Gu, Elliott Ash, Richard Hahnloser), ACL Main Conference (2022)

## Previous Year's Projects (3)

Other projects that are likely to get published, e.g.:

1. analysis of bias towards immigrants in the early 1900s using old newspapers.
2. an audio/text analysis of central bank speeches and inflation beliefs.
3. partisan tweet generator that responds in the style of a Republican or Democrat.

Incomplete projects that could be taken over this term:

1. system for predicting judicial decisions based on the submitted briefs
2. causal analysis using deep instrumental variables of what arguments in judicial opinions increase citations
3. grammarly or legal language
4. language debiaser using random permutation of entities

## Project Topics and First Steps

- Picking a topic:
  - You are welcome to come up with your own topic. We will provide feedback on that.
  - We have a list of suggested topics with project advisors.
  - I can also provide advice about which of these topics is a good fit based on team interests and skills.
- First steps:
  - once you have formed a group, send to Jingwei a list of team members with their research experience and interests.
    - if you are interested in one or more of the suggested topics, include that in the email
  - we will then match project advisors and set up meeting