Featurization, Part 2

Elliott Ash

Bocconi 2018

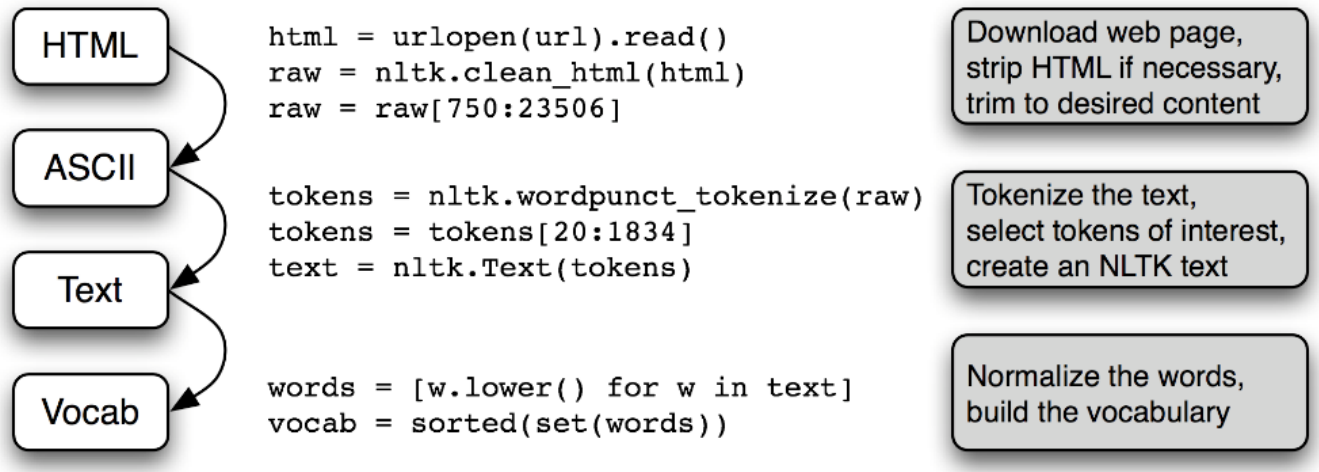
Overview

- In this lecture we build on the previous featurization steps where we created the counts of words.

Tractability vs. Predictiveness vs. Interpretability

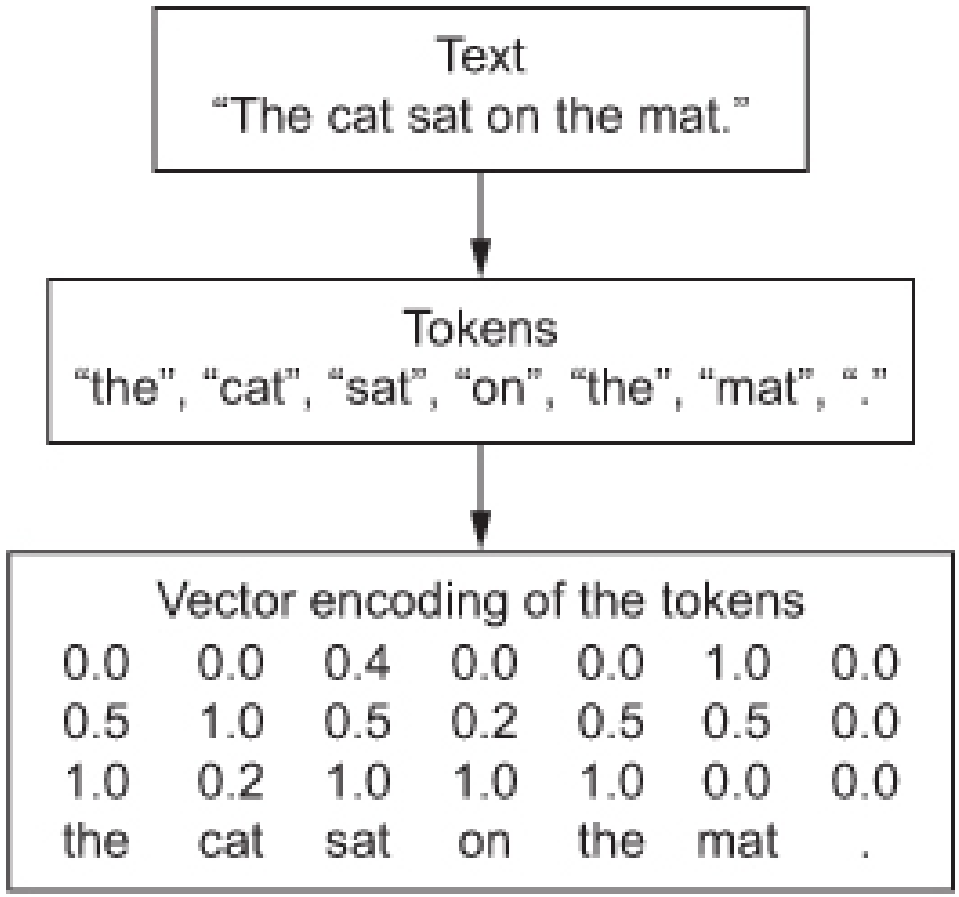
- A major goal of featurization is to produce features that are
  - **predictive** in the learning task
  - **interpretable** by human investigators
  - **tractable** enough to be easy to work with

NLP Pipeline



Source: NLTK Book, Chapter 3.

NLP Pipeline (2)



Document frequencies and term frequencies

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

- **Relative frequency**:

$$\text{Relative Frequency in document } k = \frac{\text{Term count in document } k}{\text{Total tokens in document } k}$$

Building a vocabulary

- An important featurization step is to build a vocabulary of words:
  - Compute document frequencies for all words
  - Inspect low-frequency words and determine a minimum document threshold.
    - e.g., 10 documents, or .25% of documents.
- Can also impose more complex thresholds, e.g.:
  - appears twice in at least 20 documents
  - appears in at least 3 documents in at least 5 years
- Assign numerical identifiers to tokens to increase speed and reduce disk usage.

TF-IDF Weighting

- TF/IDF: "Term-Frequency / Inverse-Document-Frequency."
- The formula for word w in document k:

$$\underbrace{\frac{\text{Count of } w \text{ in } k}{\text{Total word count of } k}}_{\text{Term Frequency}} \times \underbrace{\log\left(\frac{\text{Number of documents in } D}{\text{Count of documents containing } w}\right)}_{\text{Inverse Document Frequency}}$$

- Example:
  - A document contains 100 words, and the word appears 3 times. The TF is .03. The corpus has 100 documents, and the word appears in 10 documents. the IDF is $\log(100/10) \approx 2.3$, so the TF-IDF is $.03 \times 2.3 = .07$. Say the word appears in 90 out of 100 documents: Then its IDF is 0.105, with TF-IDF equal to .003.
- This formula up-weights relatively rare words that do not appear in all documents.
  - These words are probably more distinctive of topics or differences between documents.

TF-IDF Weighting: Code

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(min_df=0.01, # at min 1% of docs
                        max_df=0.9,  # at most 90% of docs
                        max_features=100000,
                        stop_words='english',
                        use_idf=True,
                        ngram_range=(1,3))
X_tfidf = tfidf.fit_transform(df1['snippet'])
pd.to_pickle(X_tfidf,'X_tfidf.pkl')
```

Our first word cloud

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from utils import get_docfreqs

f = get_docfreqs(df1['snippet']) # makes python dictionary
wordcloud = WordCloud().generate_from_frequencies(f)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Parts of speech tags

- Parts of speech provide useful word categories corresponding to their functions in sentences:
  - Eight major parts of speech: verb (VB), noun (NN), pronoun (PR+DT), adjective (JJ), adverb (RB), preposition (IN), conjunction (CC), and interjection (UH).
  - The Penn TreeBank POS tag set (used in most 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 most important
  - For sentiment, adjectives are most important.

POS tagging in Python

```
from nltk.tag.perceptron import PerceptronTagger
from nltk import word_tokenize
tagger = PerceptronTagger()
tokens = word_tokenize(text)
tagged_sentence = tagger.tag(tokens)
tagged_sentence
```

Visualization Example

```
from collections import Counter
from nltk import word_tokenize

def get_pos(snippet):
        tokens = word_tokenize(snippet)
    tags = [x[1] for x in tagger.tag(tokens)]
    num_nouns = len([t for t in tags if t[0] == 'N'])
    num_adj = len([t for t in tags if t[0] == 'J'])
    return num_nouns, num_adj

dfs = df1.sample(frac=.1)
dfs['nouns'], dfs['adj'] = zip(*dfs['snippet'].map(get_pos))
dfs.groupby('year')[['nouns','adj']].mean().plot()
```

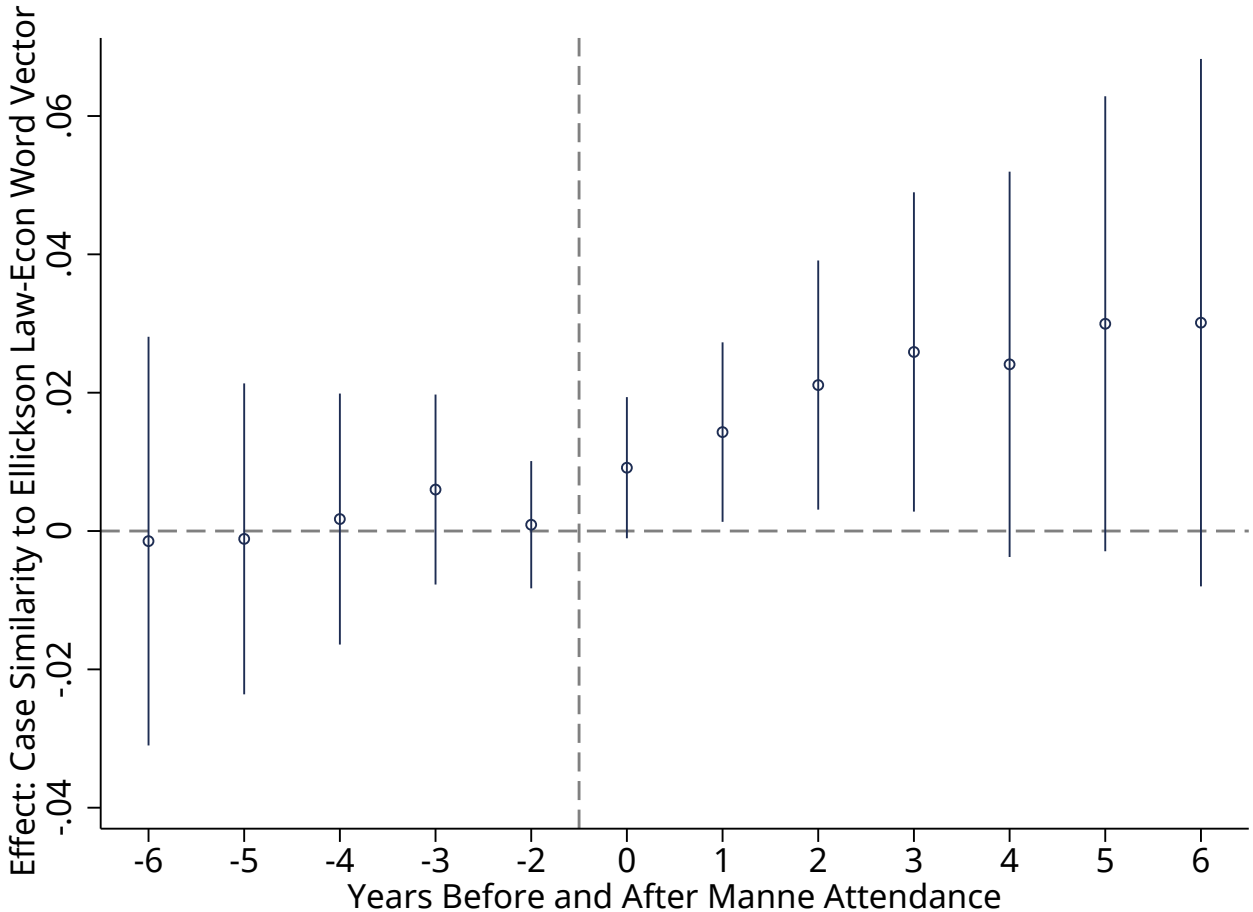
Overview of Dictionary-Based Methods

- Dictionary-based text methods use a pre-selected list of words or phrases to analyze a corpus.
- Three major categories:
  - Corpus-specific (e.g., number of times a judge says “justice” vs “efficiency”)
  - General (e.g. LIWC)
  - Sentiment Analysis

Corpus-specific words

- Sometimes counting sets of words or phrases across documents can provide useful evidence.
- Ash, Chen, and Naidu (2017):
  - We analyze the use of economics reasoning in the judiciary.
  - For example, use of the word "efficiency" or "deterrence" after attending the Manne program for judges.
    - This is a two-week intensive summer course in economics.

Impact of Economics Training on Economics Language

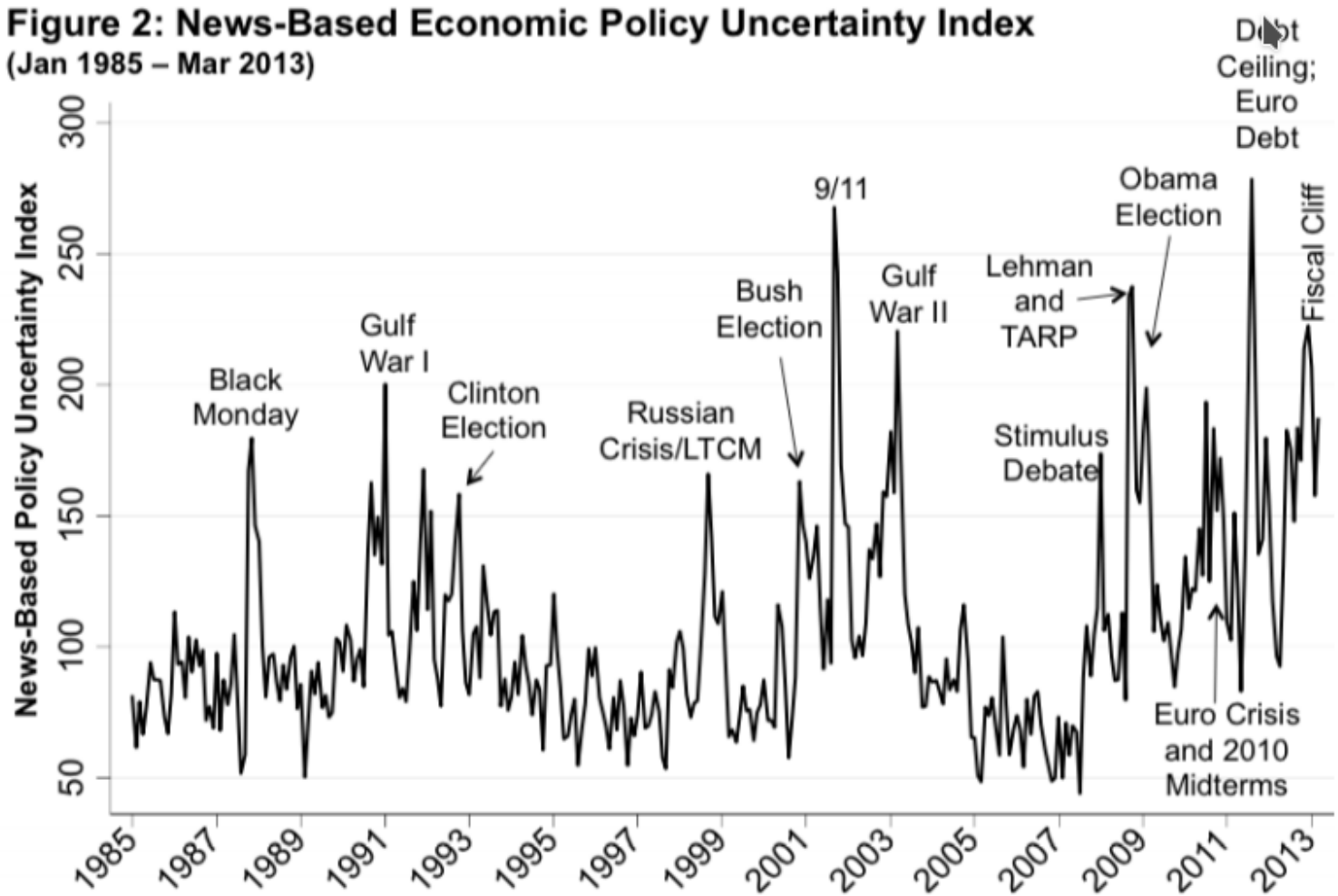


After attendance, Economics Trained Judges increase use of a selection of terms related to law and economics

Measuring uncertainty in macroeconomy

- Baker, Bloom, and Davis measure economic policy uncertainty using Boolean search of newspaper articles. (See `http://www.policyuncertainty.com/`).
- For each paper on each day since 1985, submit the following query:
  - 1. Article contains "uncertain" OR "uncertainty", AND
  - 2. Article contains "economic" OR "economy", AND
  - 3. Article contains "congress" OR "deficit" OR "federal reserve" OR "legislation" OR "regulation" OR "white house"
- Normalize resulting article counts by total newspaper articles that month.

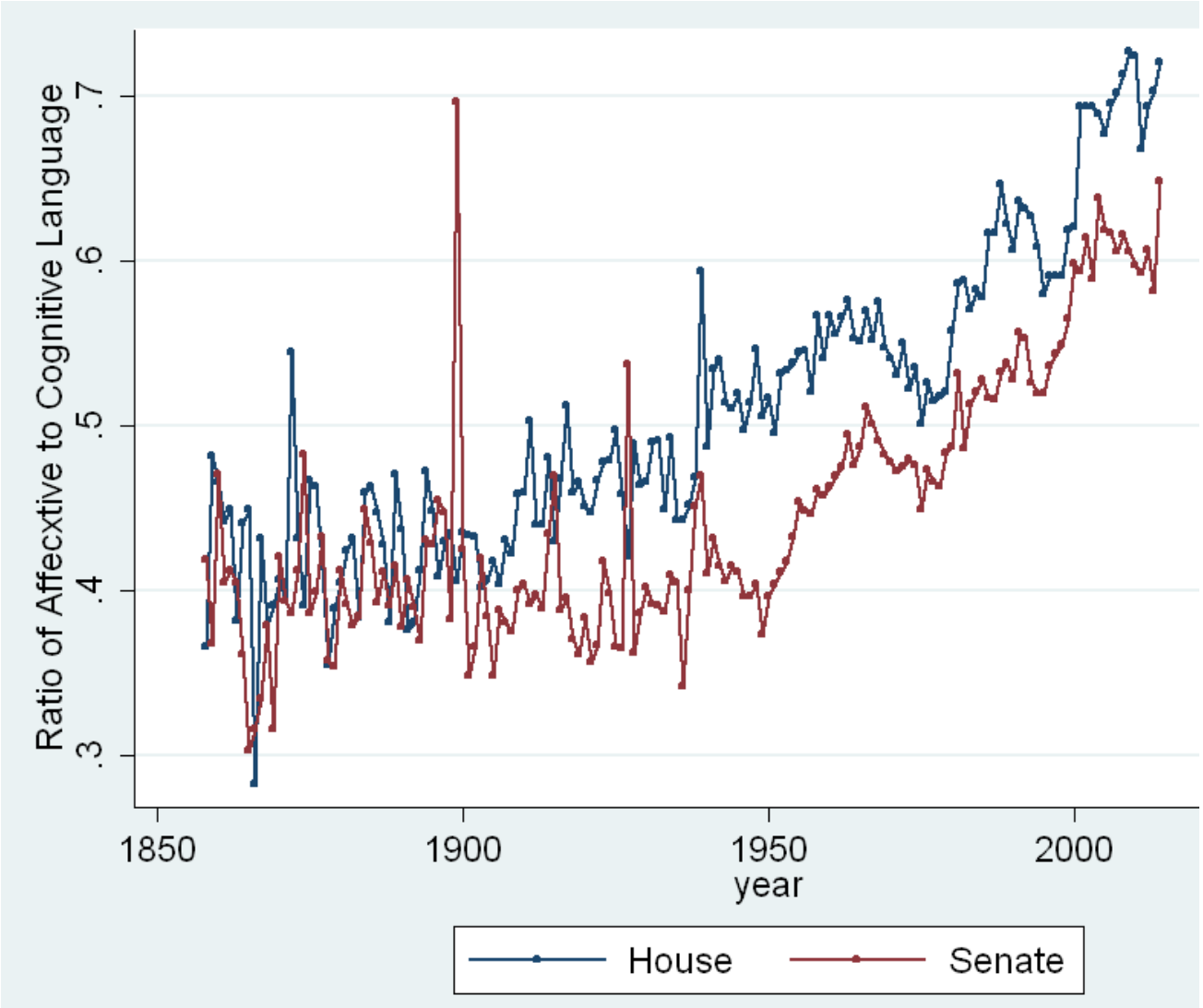
Measuring uncertainty in macroeconomy



LIWC

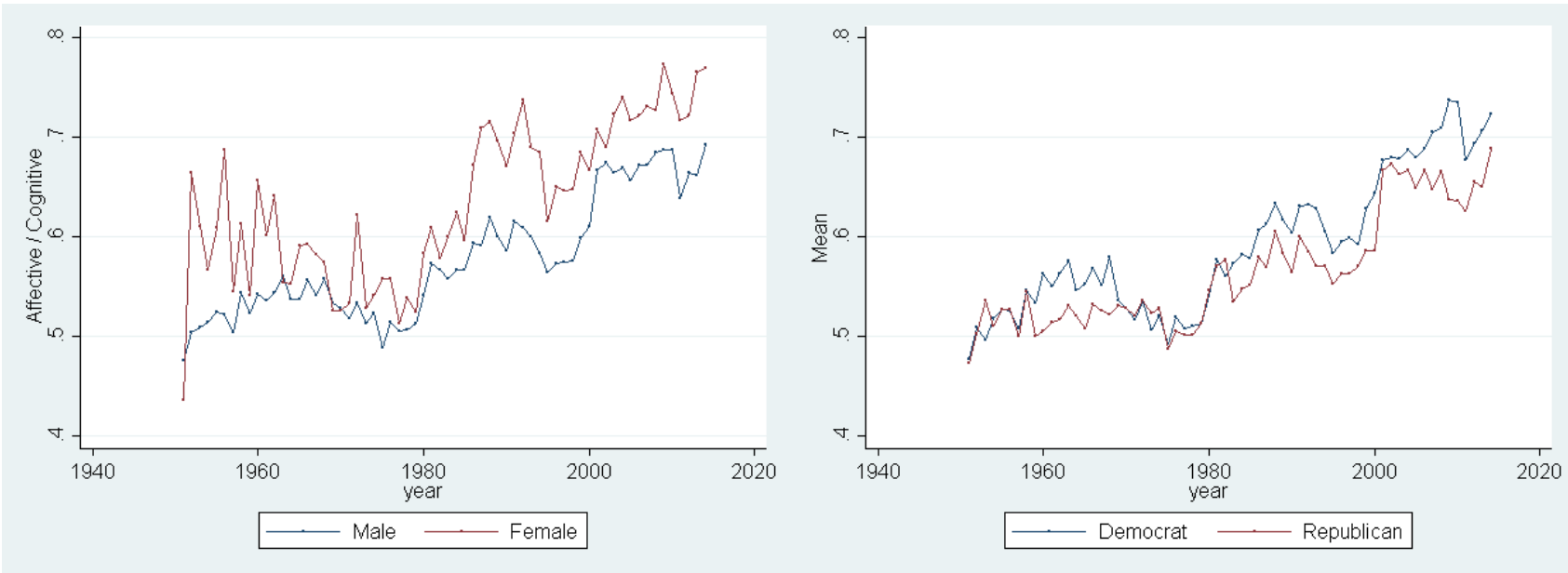
- LIWC stands for Linguistic Inquiry and Word Counts
  - Info and publications at `liwc.net`
  - Invented in 1980s, now in third version
- Word List Poster: `http://elliottash.com/wp-content/uploads/2017/07/LIWC2015-dictionary-poster.pdf`

Emotive vs. Cognitive Processing in U.S. Congress



Source: Ash and Loewen (2017)

Emotive vs. Cognitive Processing in U.S. Congress



Sentiment Analysis in Python

- The vader class in nltk provides positive, negative, and neutral scores for a document, and a composite score that combines all three.
  - vader works best on raw text – capitalization and punctuation are used in the calculus.
- Designed for online writing – hard to say how well it works on legal text, for example.

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
polarity = sid.polarity_scores(text)

def get_sentiment(snippet):
    return sid.polarity_scores(snippet)['compound']
dfs['sentiment'] = dfs['snippet'].apply(get_sentiment)
dfs.groupby('year')[['sentiment']].mean().plot()
```

Limitations of sentiment analysis

I'd hate to be the president

Limitations of sentiment analysis

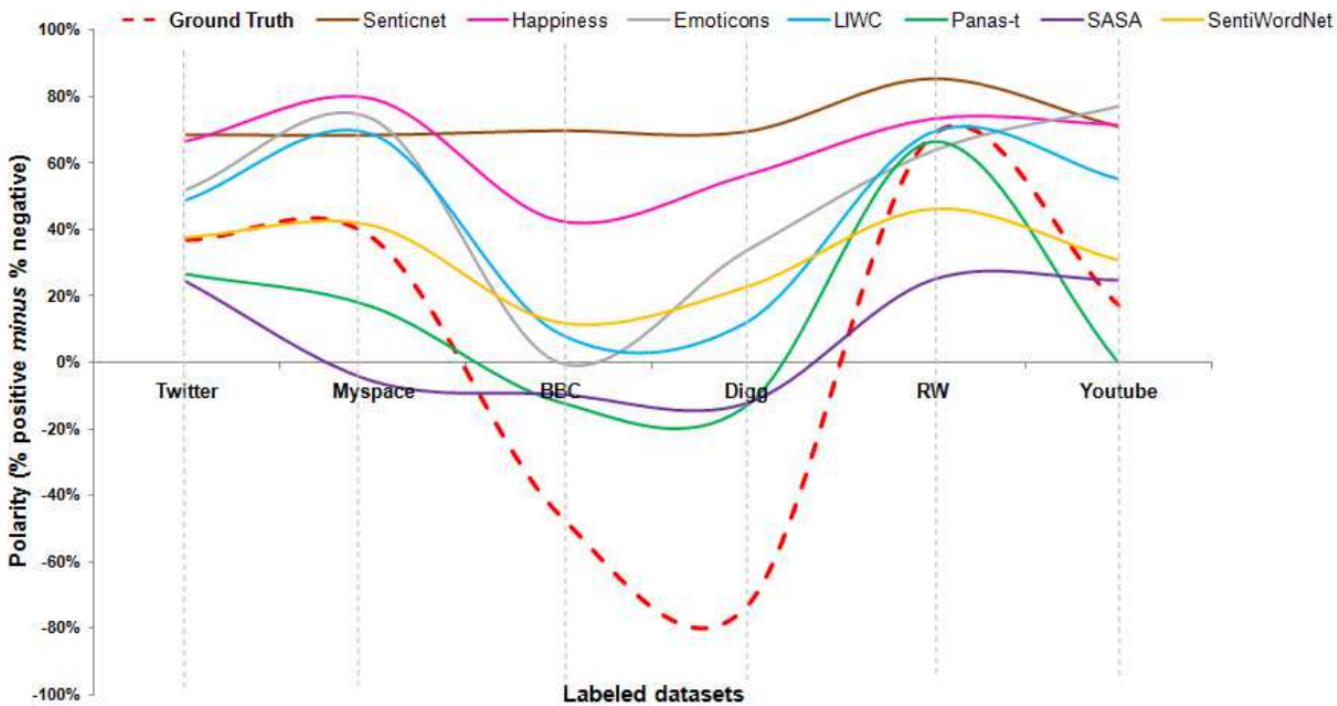
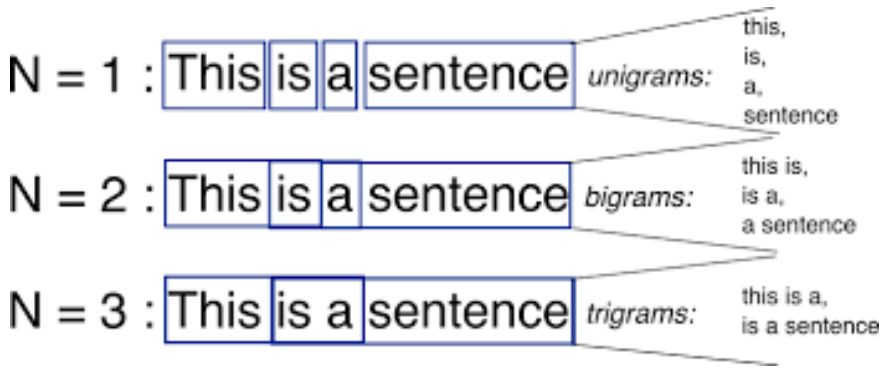


Figure 2: Polarity of the eight sentiment methods across the labeled datasets, indicating that existing methods vary widely in their agreement.

What are N-grams

- N-grams are phrases, sequences of words up to length N.
  - bigrams, trigrams, quadgrams, etc.




```
from nltk import ngrams
grams = []
for n in range(2,4):
    grams += list(ngrams(tokens,n))
Counter(grams).most_common()[:8]
```

N-grams and high dimensionality

- N-grams will blow up your feature space:
  - filtering out low-doc-frequency n-grams is necessary.
- Google experts say that a feature space with $P = 20,000$ will work well for descriptive and prediction tasks.
  - I would recommend building a vocabulary of 50K, then using feature selection (Lecture 9) to get down to 20K.

Collocations

- Conceptually, the goal of including n-grams is to featurize **collocations**:
  - Non-compositional: the meaning is not the sum of the parts (kick+the+bucket≠"kick the bucket")
  - Non-substitutable: cannot substitute components with synonyms ("fast food"≠"quick food")
  - Non-modifiable: cannot modify with additional words or grammar: (e.g., "kick around the bucket", "kick the buckets")

Point-wise mutual information

- A good way to filter N-grams is by point-wise mutual information:

$$\begin{aligned}\mathrm{PMI}(w_1, w_2) &= \frac{\Pr(w_1, w_2)}{\Pr(w_1)\Pr(w_2)} \\ &= \frac{\text{Prob. of collocation, actual}}{\text{Prob. of collocation, if independent}}\end{aligned}$$

  where w_1 and w_2 are words in the vocabulary, and w_1, w_2 is the N-gram $w_1\_w_2$.

- It ranks words by how often they collocate, relative to how often they occur apart.

- Disadvantage:
  - Rare words that appear together once or twice will have high PMI.
  - Address this with minimum frequency thresholds.

Geometric Mean: PMI for N>2

- PMI can be generalized to arbitrary N as the geometric mean of the probabilities:

$$\frac{\Pr(w_1, ..., w_N)}{\prod_{i=1}^{n} \sqrt[n]{\Pr(w_i)}}$$

- E.g., for trigrams:

$$\frac{\Pr(w_1, w_2, w_3)}{\sqrt[3]{\Pr(w_1)\Pr(w_2)\Pr(w_3)}}$$

- The n-root normalizer is not necessary (it does not change the ranking), but makes scores for bigrams/trigrams/quadgrams/etc. more comparable.

Computing Geometric Mean with N-gram Frequencies

- Probability of a token is the frequency in the corpus:

$$\Pr(w_1) = \frac{\text{Count}(w_1)}{\sum_{i=1}^{P} \text{Count}(w_i)}$$

  where P is vocabulary size.

- Let $f_i = \text{Count}(w_i)$ and $F = \sum_{i=1}^{P} f_i$. Then we have

$$\text{PMI}(w_1, w_2) = \frac{\Pr(w_1, w_2)}{\Pr(w_1)\Pr(w_2)} = \frac{\frac{f_{12}}{F}}{\frac{f_1}{F} \cdot \frac{f_2}{F}} = \frac{1}{F}\frac{f_{12}}{f_1 f_2}$$

- Note that the leading $\frac{1}{F}$ does not affect the ranking of bigrams, and cancels out with the geometric mean formula:

$$\text{gmean}(w_1, w_2) = \frac{\Pr(w_1, w_2)}{\sqrt{\Pr(w_1)\Pr(w_2)}} = \frac{\frac{f_{12}}{F}}{\sqrt{\frac{f_1}{F} \cdot \frac{f_2}{F}}} = \frac{f_{12}}{\sqrt{f_1 f_2}}$$

- Similarly, it cancels out for $N > 2$.
- Therefore PMI can be computed directly from term counts (rather than frequencies).

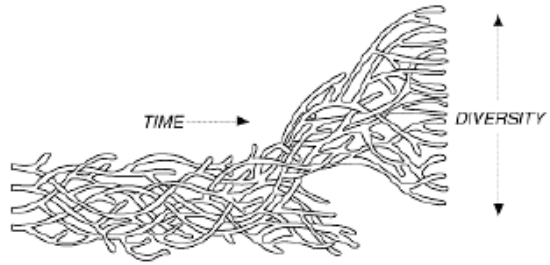
Constructing Memes from Informative Phrases

- Legal terminology:
  - A: Adjective, N: Noun, V: Verb, P: Preposition, D: Determinant, C: Conjunction.
  - 2-grams: AN, NN, VN, VV, NV, VP.
  - 3-grams: NNN, AAN, ANN, NAN, NPN, VAN, VNN, AVN, VVN, VPN, ANV, NVV, VDN, VVV, NNV, VVP, VAV, VVN, NCN, VCV, ACA, PAN.
  - 4-grams: NCVN, ANNN, NNNN, NPNN, AANN, ANNN, ANPN, NNPN, NPAN, ACAN, NCNN, NNCN, ANCN, NCAN, PDAN, PNPN, VDNN, VDAN, VVDN.

What does it get you?

- `Beyond a reasonable doubt (preposition, article, adjective, noun)`
- `Earned income tax credit (adjective, noun, noun, noun)`
- I have included an optimized implementation in code_06_features-2.py.

Scoring Memetic Phrases (Chen et al 2017)



$$P_m = \frac{d_{m \to m}}{d_{\to m} + \delta} / \frac{d_{m \to \not m} + \delta}{d_{\to \not m} + \delta}$$

- $d_{m \to m}$ =# of cases with gram m, and cite $\geq$1 case with m
- $d_{\to m}$ =# of cases which cite $\geq$1 case with m
- $d_{m \to \sim m}$ =# of cases with m that do not cite any other case with m
- $d_{\to \sim m}$=# of cases which do not cite any other case with m
- δ, noise factor to account for non-citing cases

Memetic Phrases in Law and Economics

- Ash,Chen,Naidu (2018) find that "deterrence_theory" and "capital" show high memetic tendencies:
  - Judges learn to use these terms from each other
  - These terms tend to cross topic boundaries
  - Transmission is stronger between judges in the same political party.

Dependencies and Sentence Structure

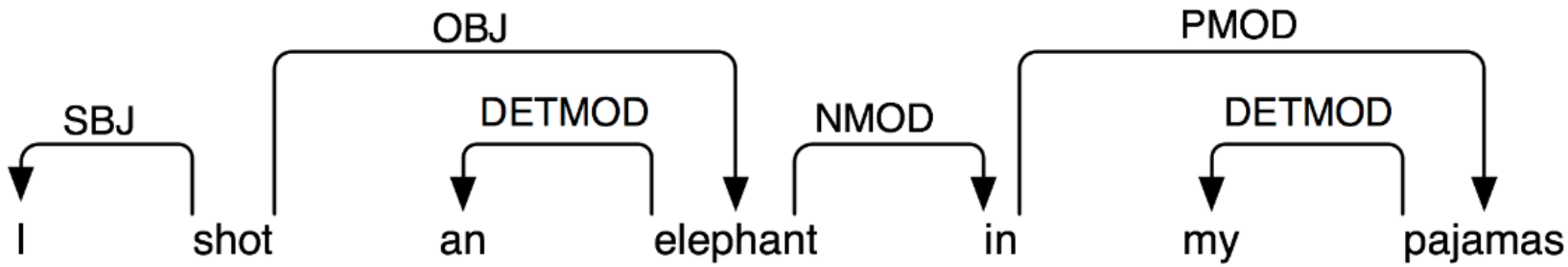
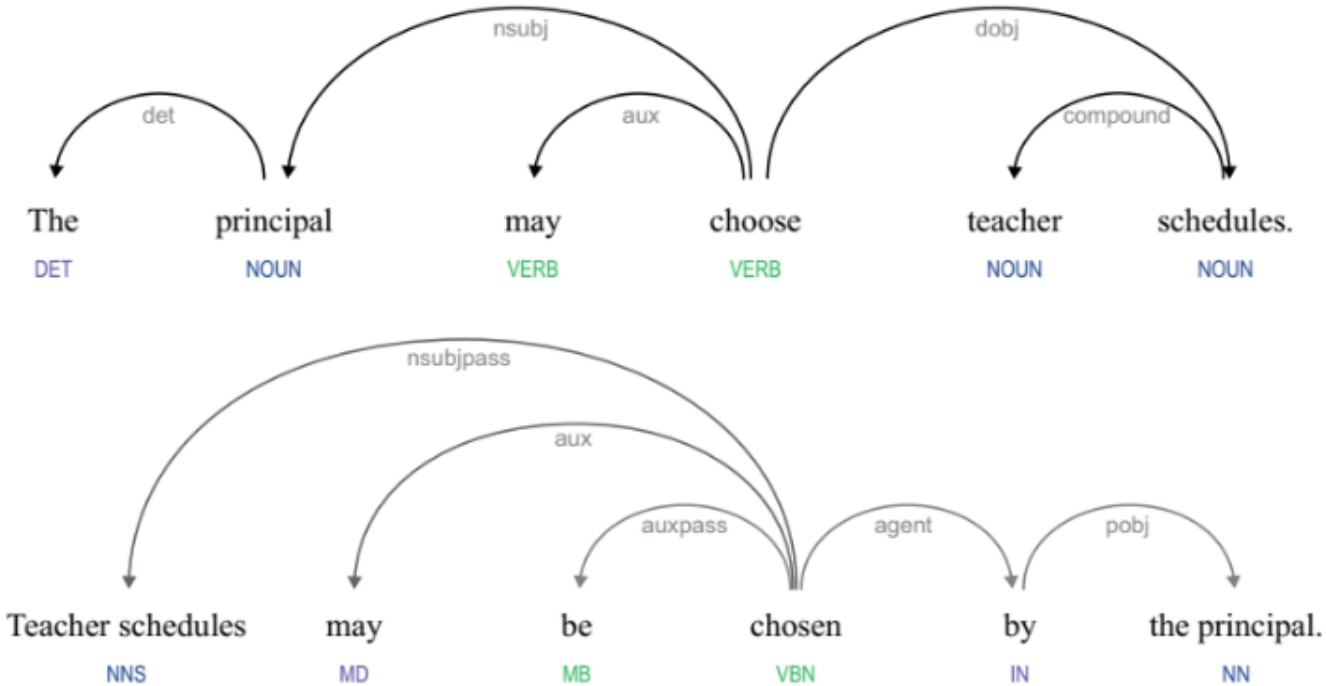


***Figure 5.1**: Dependency Structure: arrows point from heads to their dependents; labels indicate the grammatical function of the dependent as subject, object or modifier.*

Parsing Sentences in spaCy

```
import spacy
nlp = spacy.load('en')
doc = nlp(text)
for sent in doc.sents:
    print(sent.root)
    print([(w, w.dep_) for w in sent.root.children])
    print()
```

Syntactic Parsers



- The parser transforms sentences into parse trees, which represent the relations between words in a recursive hierarchical structure.

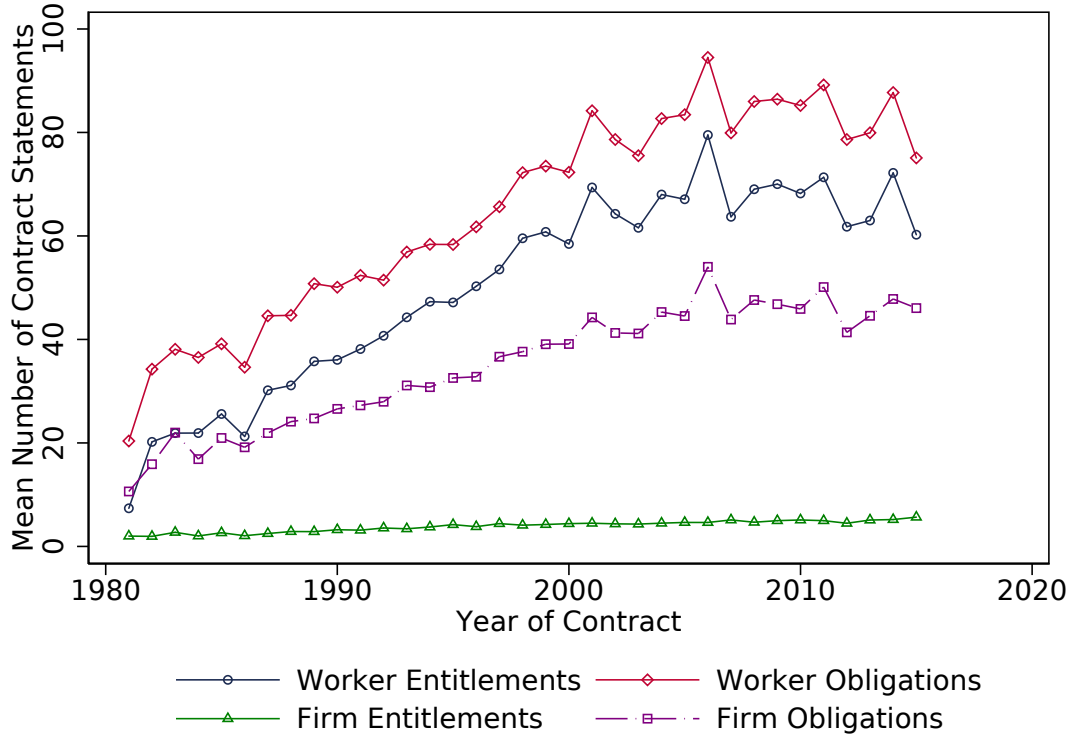
Application: Union Contract Language

- Ash, MacLeod, and Naidu (2017) approach to measuring the features of contracts:
  - Extract the subject, modal, and associated (action) verb.
  - Subject is assigned (sometimes probabilistically) to one of four agent categories:
    - worker, union, owner, and manager.
  - Modal verbs:
    - strict (*shall*, *will*, *must*)
    - permissive (*may*, *can*)

Contract Statement Logic

Categorization Logic	Examples
Obligations	
Positive, Strict Modal, Active Verb	shall be, shall provide, shall include, shall notify, shall continue
Positive, Strict Modal, Obligation Verb	shall be required, shall be expected, shall be obliged
Positive, Non-Modal, Obligation Verb	is required, is expected
Prohibitions	
Negative, Any Modal, Active Verb	shall not exceed, shall not use, shall not apply, shall not discriminate
Negative, Permission Verb	shall not be allowed, is not permitted
Positive, Strict Modal, Constraint Verb	shall be prohibited, shal be restricted
Permissions	
Positive, Non-Modal, Permission Verb	is allowed, is permitted, is authorized
Positive, Strict Modal, Permission Verb	shall be allowed, shall be permitted
Positive, Permissive Modal, Active Verb	may be, may request, may use, may require, may apply
Negative, Any Modal, Constraint Verb	shall not be restricted, shall not be prohibited
Entitlements	
Strict Modal, Passive Verb	shallbe paid, shall be given, shall not be discharged
Positive, Strict Modal, Entitlement Verb	shall have, shall receive, shall retain
Negative, Any Modal, Obligation Verb	may not be required

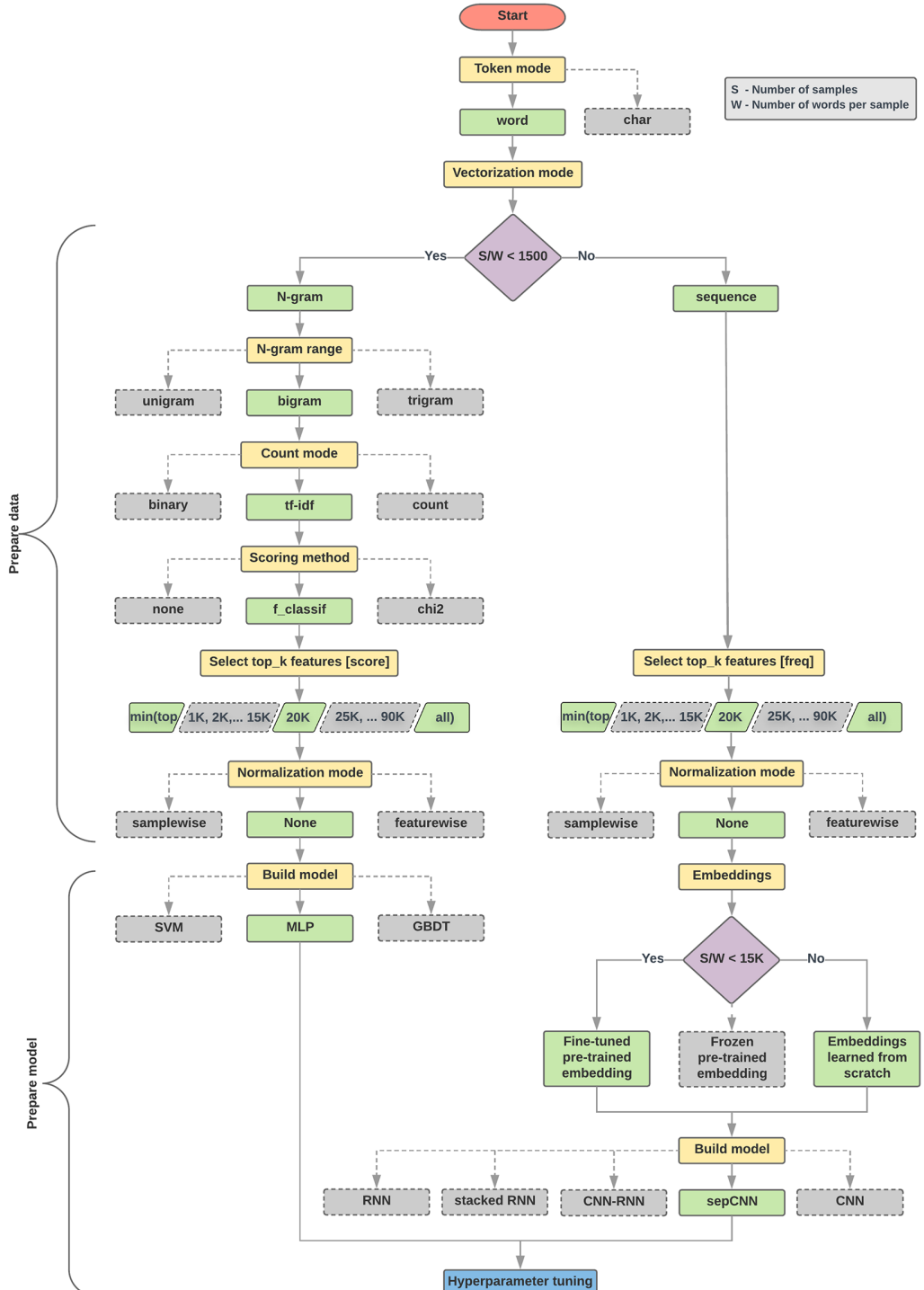
Obligations and Entitlements over Time, By Agent

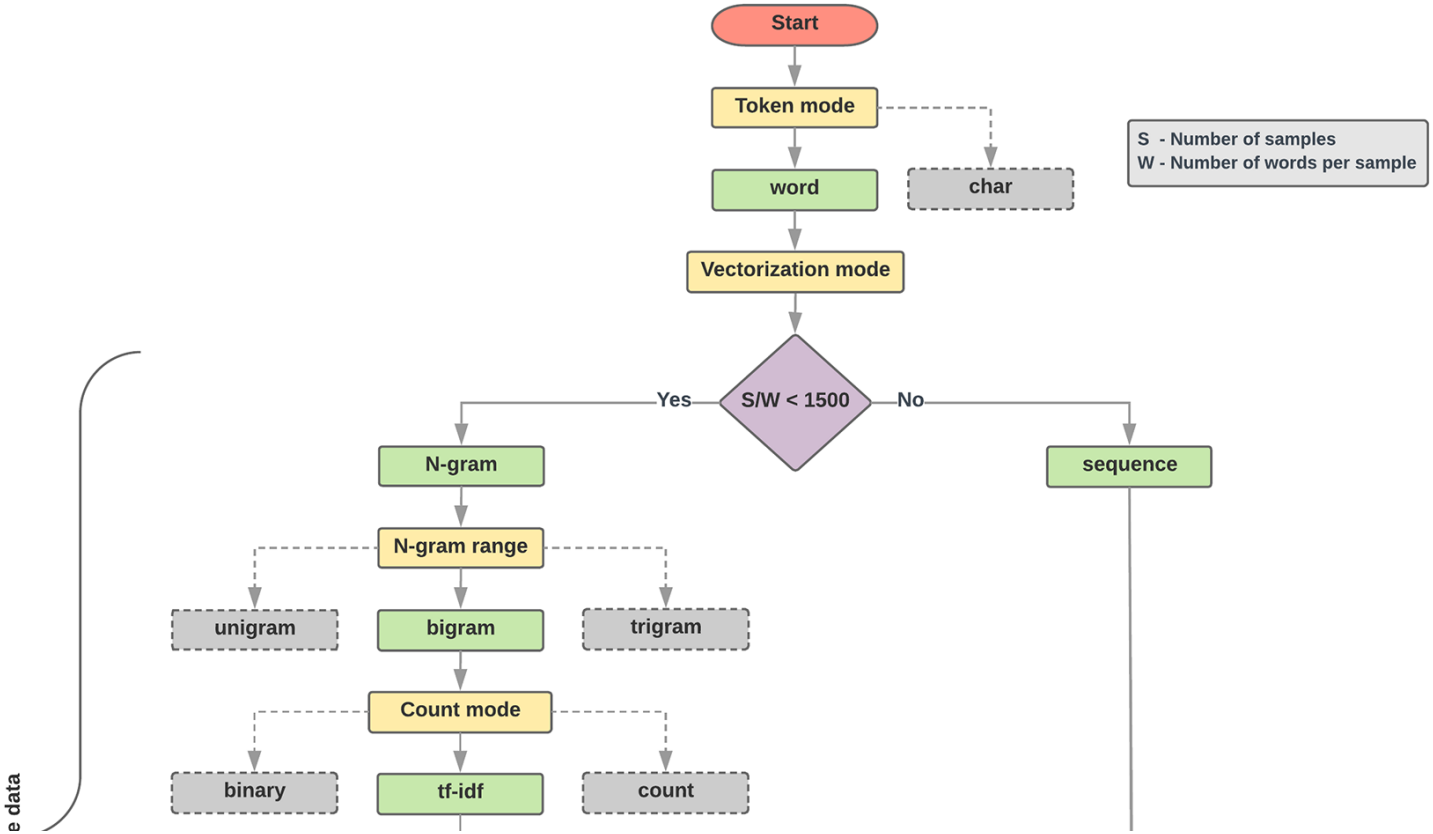


- Workers and firms have gotten more obligations over time.
- Only workers have gotten more entitlements over time.

Additional features topics

- See the NLTK book or google them:
  - Co-reference resolution
  - Named entity recognition
  - Relation extraction


Start
Token mode
word
char
S - Number of samples
W - Number of words per sample
Vectorization mode
S/W < 1500
Yes
No
N-gram
N-gram range
unigram
bigram
trigram
Count mode
binary
tf-idf
count
Scoring method
none
f_classif
chi2
Select top_k features [score]
min(top, 1K, 2K,... 15K, 20K, 25K, ... 90K, all)
Normalization mode
samplewise
None
featurewise
Build model
SVM
MLP
GBDT
sequence
Select top_k features [freq]
min(top, 1K, 2K,... 15K, 20K, 25K, ... 90K, all)
Normalization mode
samplewise
None
featurewise
Embeddings
S/W < 15K
Yes
No
Fine-tuned pre-trained embedding
Frozen pre-trained embedding
Embeddings learned from scratch
Build model
RNN
stacked RNN
CNN-RNN
sepCNN
CNN
Hyperparameter tuning
Prepare data
Prepare model



- Google Developers recommend **tf-idf-weighted bigrams** as a baseline specification for text classification tasks.
  - ideal for fewer, longer documents.
- With more numerous, shorter documents (rows / doclength $>$ 1500), better to use an embedded sequence.
  - To be described in Lecture 14.