# Sequencing Legal DNA: NLP for Law and Political Economy

## 2. Tokenization

**Instructions before we begin:**
(1) Turn on video and set audio to mute
(2) In Participants panel, set zoom name to "Full Name, School / Degree"
(ex: "Leon Smith, ETH Data Science Msc")
(3) If this is your first lecture, say "hi" in the chat

# Online Lecture Norms

**Let's make the most of online learning!**

- Live attendance at lectures is required.
- Keep video on if connection allows.
- Stay muted when not talking.
- To make questions or comments, use the "raise hand" function.

# Homework

- First homework is due tomorrow by midnight.
- Submit on EduFlow (reachable from moodle, link will be on syllabus).

# TA Session

- Any feedback on the first TA session?
  - video is linked on syllabus.
- Second TA session is this Friday, 1230h-1330h
  - go over week 1 homework
  - go over week 2 notebook
  - can ask questions in advance using TA Q&A page (on syllabus).

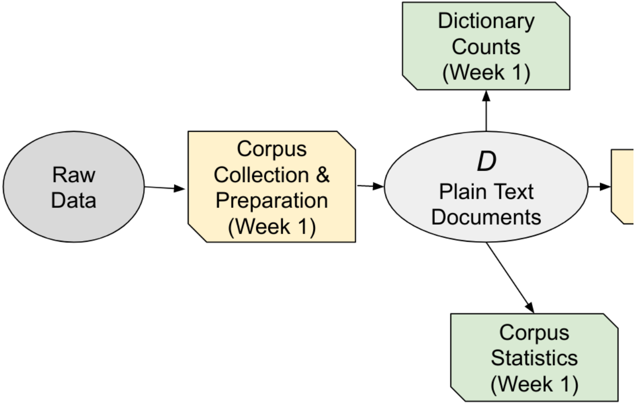
# Final Assignment Info

- For those not doing a project, there is a final assignment distributed a week or two after class ends.
    - Questions will be based on the slides and some new readings.
    - You will have three days to do it.
    - Will distribute practice questions beforehand.

# Week 2 Q&A Page

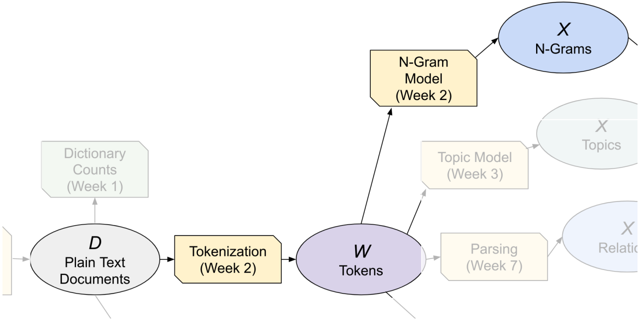
http://bit.ly/NLP-QA02

# Last Week


Dictionary
Counts
(Week 1)
Raw
Data
Corpus
Collection &
Preparation
(Week 1)
D
Plain Text
Documents
Corpus
Statistics
(Week 1)

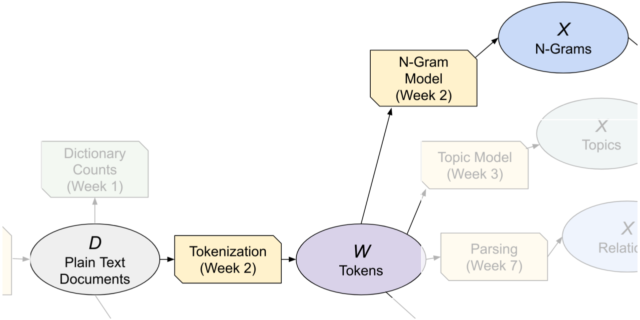
# Practice with Dictionary Methods

# Today



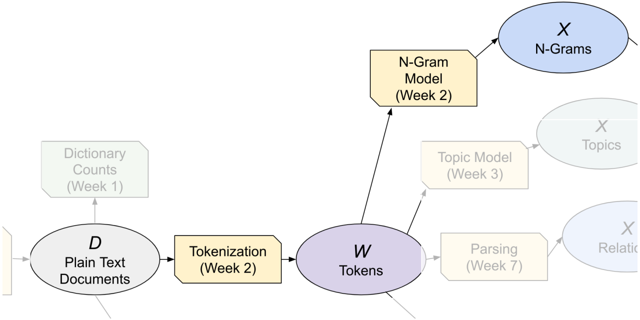

- Input:
  - A set of documents (e.g. text files), $D$.

# Today




- Input:
  - A set of documents (e.g. text files), $D$.
- Output (tokens):
  - A sequence, $W$, containing a list of tokens – words or word pieces for use in natural language processing

# Today




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

# Goals of Tokenization

To summarize: A major goal of tokenization is to produce features that are

- **predictive** in the learning task
- **interpretable** by human investigators
- **tractable** enough to be easy to work with

# Goals of Tokenization

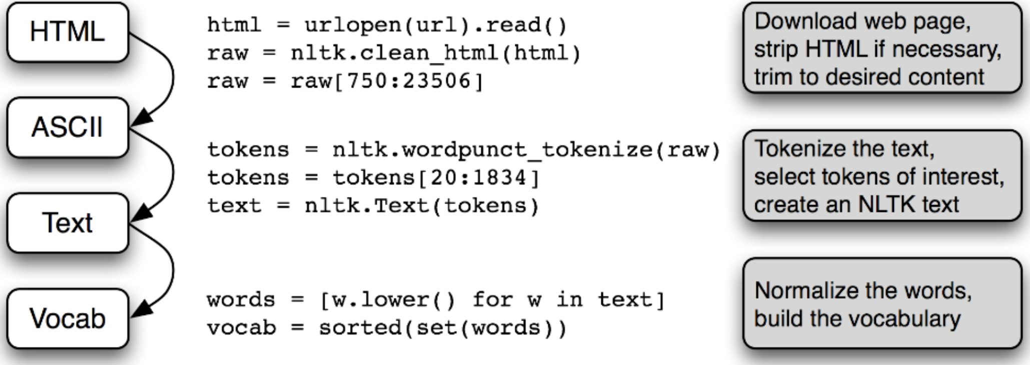
To summarize: A major goal of tokenization is to produce features that are

- **predictive** in the learning task
- **interpretable** by human investigators
- **tractable** enough to be easy to work with

**Two broad approaches:**

1. convert documents to vectors, usually frequency distributions over pre-processed n-grams.
2. convert documents to sequences of tokens, for inputs to sequential models.

# A Standard Tokenization Pipeline




Source: NLTK Book, Chapter 3.

# Segmenting paragraphs/sentences

- Many tasks should be done on sentences, rather than corpora as a whole.
  - NLTK and spaCy do a good (but not perfect) job of splitting sentences, while accounting for periods on abbreviations, etc.
  - spaCy is slower but significantly better.

# Segmenting paragraphs/sentences

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

# Pre-processing

- An important piece of the "art" of text analysis is deciding what data to throw out.
  - Uninformative data add noise and reduce statistical precision.
  - They are also computationally costly.

# Pre-processing

- An important piece of the "art" of text analysis is deciding 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).
  - some features are more interpretable: "judge has" / "has discretion" vs "judge has discretion".

# Capitalization

- Removing capitalization is a standard corpus normalization technique
  - usually the capitalized/non-capitalized version of a word are equivalent – e.g. words showing up capitalized at beginning of sentence
  - $\rightarrow$ capitalization not informative.

# Capitalization

- Removing capitalization is a standard corpus normalization technique
  - usually the capitalized/non-capitalized version of a word are equivalent – e.g. words showing up capitalized at beginning of sentence
  - $\rightarrow$ capitalization not informative.
- Also: what about "the first amendment" versus "the First Amendment"?

# Capitalization

- Removing capitalization is a standard corpus normalization technique
  - usually the capitalized/non-capitalized version of a word are equivalent – e.g. words showing up capitalized at beginning of sentence
  - $\rightarrow$ capitalization not informative.
- Also: what about "the first amendment" versus "the First Amendment"?
  - Compromise: include capitalized version of words not at beginning of sentence.

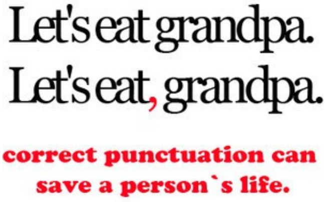
# Capitalization

- Removing capitalization is a standard corpus normalization technique
  - usually the capitalized/non-capitalized version of a word are equivalent – e.g. words showing up capitalized at beginning of sentence
  - $\rightarrow$ capitalization not informative.
- Also: what about "the first amendment" versus "the First Amendment"?
  - Compromise: include capitalized version of words not at beginning of sentence.
- For some tasks, capitalization is important
  - needed for sentence splitting, part-of-speech tagging, syntactic parsing, and semantic role labeling.

# Capitalization

- Removing capitalization is a standard corpus normalization technique
  - usually the capitalized/non-capitalized version of a word are equivalent – e.g. words showing up capitalized at beginning of sentence
  - $\rightarrow$ capitalization not informative.
- Also: what about "the first amendment" versus "the First Amendment"?
  - Compromise: include capitalized version of words not at beginning of sentence.
- For some tasks, capitalization is important
  - needed for sentence splitting, part-of-speech tagging, syntactic parsing, and semantic role labeling.
  - For sequence data, e.g. language modeling – huggingface tokenizer takes out capitalization but then add a special "capitalized" token before the word.

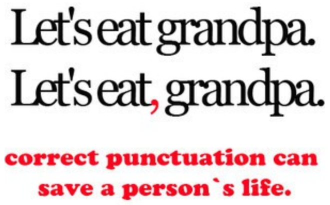
# Punctuation




Source: Chris Bail text data slides.

Inclusion of punctuation depends on your task:

# Punctuation

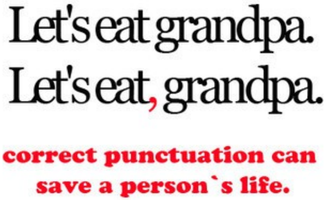



Source: Chris Bail text data slides.

Inclusion of punctuation depends on your task:

- if you are vectorizing the document as a bag of words or bag of n-grams, punctuation won't be needed.

# Punctuation




Source: Chris Bail text data slides.

Inclusion of punctuation depends on your task:

- if you are vectorizing the document as a bag of words or bag of n-grams, punctuation won't be needed.
- like capitalization, punctuation is needed for annotations (sentence splitting, parts of speech, syntax, roles, etc)
  - also needed for language models.

# Numbers

- for classification using bag of words:
  - can drop numbers, or replace with special characters

# Numbers

- for classification using bag of words:
  - can drop numbers, or replace with special characters
- for language models:
  - just treat them like letters.
  - GPT-3 can solve math problems.

# Drop 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 | | | | | |

# Drop 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 | | | | | |

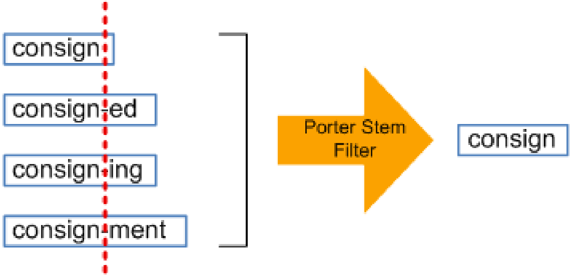
- What about "not guilty"?
- Legal "memes" often contain stopwords:
  - "beyond a reasonable doubt"
  - "with all deliberate speed"

# Drop 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 "memes" often contain stopwords:
  - "beyond a reasonable doubt"
  - "with all deliberate speed"
- can drop stopwords by themselves, but keep them as part of phrases.
- can filter out words and phrases using part-of-speech tags (later).

# 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"

# Brainstorming Activity: How to use non-word features

Depending on the first letter of your last name, do one of the following tasks.
Outline a **social-science analysis or dimension of language** that:

- A-F – can be measured by capitalization.
- G-L – can be measured by punctuation.
- M-R – would change depending on the use of stopwords.
- S-Z – would change depending on the use of stemming/lemmatizing.

Put your answer in the menti (see chat). Multiple answers welcome. Start answer with first letter of last name, .e.g "D - [answer]".

# Tokens

The most basic unit of representation in a text.

- characters: documents as sequence of individual letters {h,e,l,l,o, ,w,o,r,l,d}

# Tokens

The most basic unit of representation in a text.

- characters: documents as sequence of individual letters {h,e,l,l,o, ,w,o,r,l,d}
- words: split on white space {hello, world}

# Tokens

The most basic unit of representation in a text.

- characters: documents as sequence of individual letters $\{$h,e,l,l,o, ,w,o,r,l,d$\}$
- words: split on white space $\{$hello, world$\}$
- n-grams: learn a vocabulary of phrases and tokenize those: "ETH Zurich $\rightarrow$ ETH_Zurich"

# Tokens

The most basic unit of representation in a text.

- characters: documents as sequence of individual letters {h,e,l,l,o, ,w,o,r,l,d}
- words: split on white space {hello, world}
- n-grams: learn a vocabulary of phrases and tokenize those: "ETH Zurich $\rightarrow$ ETH_Zurich"
- what else?

# Sub-Word Units

- The standard tokenization approach in deep learning (i.e. language models) uses sub-word information.
- Tokenizers like **SentencePiece** do tokenizing at the character level, with white space and punctuation treated equivalently to alphanumeric characters.
  - requires lots of data/compute but learns word endings etc
- We will come back to this starting week 5.

# 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 token appears.
- **Term counts**: number of total appearances of a token in corpus.

# Counts and frequencies

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

- 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.

# 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

# 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 $k$:

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

# TF-IDF Weighting

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

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

- The 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.
  - Example: A document contains 100 words, and the word appears 3 times in the document. 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 for this document is $.03 \times 2.3 = .07$. Say the word appears in 90 out of 100 documents: Then the IDF is 0.105, with TF-IDF for this document equal to .003.

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

- `corpus` is a sequence of strings, e.g. pandas data-frame columns.

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

- **`corpus`** is a sequence of strings, e.g. pandas data-frame columns.
- pre-processing options: strip accents, lowercase, drop stopwords,

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

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

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

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

# scikit-learn's TfidfVectorizer

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(corpus)
<4x9 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

- `corpus` is a sequence of strings, e.g. pandas data-frame columns.
- pre-processing options: strip accents, lowercase, drop stopwords,
- n-grams: can produce phrases up to length n (words or characters).
- vocab options: min/max frequency, vocab size
- post-processing: binary, l2 norm, (smoothed) idf weighting, etc

# Other Transformations?

- e.g., Kelly et al (2019) suggest that including indicators for whether a phrase appears in a document (rather than the count) is often independently predictive.

# Other Transformations?

- e.g., Kelly et al (2019) suggest that including indicators for whether a phrase appears in a document (rather than the count) is often independently predictive.
- Could add log counts, quadratics in counts, etc.
- Could also add pairwise interactions between word counts/frequencies.

# Other Transformations?

- e.g., Kelly et al (2019) suggest that including indicators for whether a phrase appears in a document (rather than the count) is often independently predictive.
- Could add log counts, quadratics in counts, etc.
- Could also add pairwise interactions between word counts/frequencies.
- These often are not done much because of the dimensionality problem.

# Other Transformations?

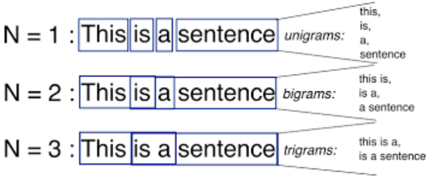
- e.g., Kelly et al (2019) suggest that including indicators for whether a phrase appears in a document (rather than the count) is often independently predictive.
- Could add log counts, quadratics in counts, etc.
- Could also add pairwise interactions between word counts/frequencies.
- These often are not done much because of the dimensionality problem.
  - could use feature selection or principal components to deal with that.

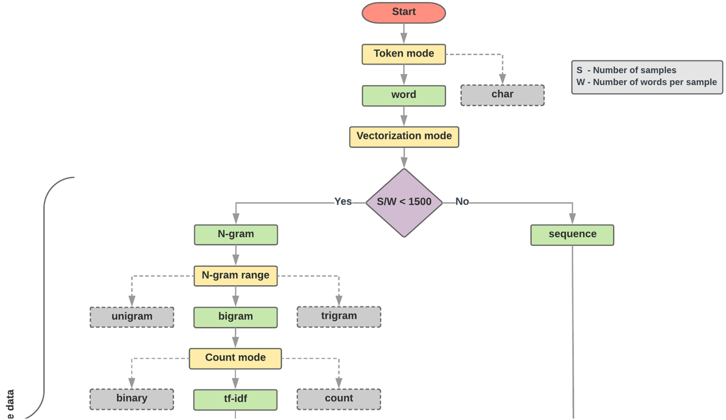
# Other Transformations?

- e.g., Kelly et al (2019) suggest that including indicators for whether a phrase appears in a document (rather than the count) is often independently predictive.
- Could add log counts, quadratics in counts, etc.
- Could also add pairwise interactions between word counts/frequencies.
- These often are not done much because of the dimensionality problem.
  - could use feature selection or principal components to deal with that.
  - for machine learning, could use SVM with a polynomial kernel.

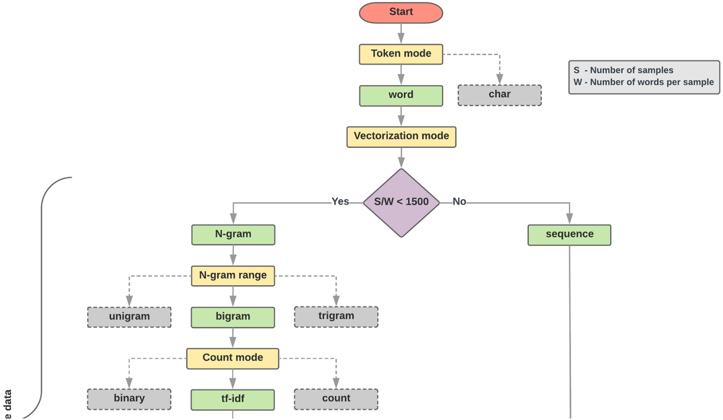
# What are N-grams

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






- Google Developers recommend **tf-idf-weighted bigrams** as a baseline specification for text classification tasks.
  - ideal for fewer, longer documents.




- 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 (starting Week 5).

# N-grams and high dimensionality

- N-grams will blow up your feature space:
  - filtering out uninformative n-grams is necessary.

# N-grams and high dimensionality

- N-grams will blow up your feature space:
  - filtering out uninformative n-grams is necessary.
- Google Developers say that a feature space with $P = 20,000$ will work well for descriptive and prediction tasks.
  - I have gotten good performance with 10K or even 2K features.
  - For supervised learning tasks, a decent baseline is to build a vocabulary of 60K, then use feature selection to get down to 10K.

# Feature selection using univariate comparisions

- $\chi^2$ is a very fast feature selection routine for classification tasks
  - features must be non-negative
  - works on sparse matrices
  - works on multi-class problems

# Feature selection using univariate comparisions

- $\chi^2$ is a very fast feature selection routine for classification tasks
  - features must be non-negative
  - works on sparse matrices
  - works on multi-class problems
- With negative predictors:
  - use `f_classif`.

# Feature selection using univariate comparisions

- $\chi^2$ is a very fast feature selection routine for classification tasks
  - features must be non-negative
  - works on sparse matrices
  - works on multi-class problems
- With negative predictors:
  - use `f_classif`.
- For regression tasks:
  - use `f_regression or OLS coefficients.`

# De-Meaning or Residualizing Variables

- For f_classif and f_regression, can de-mean predictors by groups, for example by year or location.
  - purges correlated information and can help model generalize to new years/locations.
  - for regression, can also de-mean the outcome.

# De-Meaning or Residualizing Variables

- For f_classif and f_regression, can de-mean predictors by groups, for example by year or location.
  - purges correlated information and can help model generalize to new years/locations.
  - for regression, can also de-mean the outcome.
- What if you want to de-mean by both year and location?
  - $\rightarrow$ take residuals from OLS regression of each variable (outcome and predictor) on the category dummies.

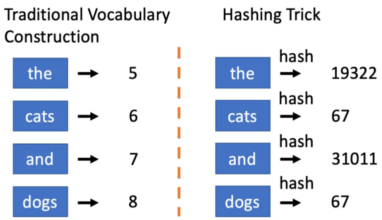
# De-Meaning or Residualizing Variables

- For f_classif and f_regression, can de-mean predictors by groups, for example by year or location.
  - purges correlated information and can help model generalize to new years/locations.
  - for regression, can also de-mean the outcome.
- What if you want to de-mean by both year and location?
  - $\rightarrow$ take residuals from OLS regression of each variable (outcome and predictor) on the category dummies.
- That is:
  - regress $Y_i = \text{FE}_1 + \text{FE}_2 + \epsilon_i$ and $x_i^w = \text{FE}_1 + \text{FE}_2 + \epsilon_i, \forall w$,
  - take residuals $\tilde{Y}_i = Y_i - \hat{Y}_i$ and $\tilde{x}_i^w = x_i^w - \hat{x}_i^w$

# De-Meaning or Residualizing Variables

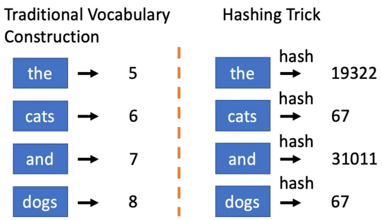
- For f_classif and f_regression, can de-mean predictors by groups, for example by year or location.
  - purges correlated information and can help model generalize to new years/locations.
  - for regression, can also de-mean the outcome.
- What if you want to de-mean by both year and location?
  - $\rightarrow$ take residuals from OLS regression of each variable (outcome and predictor) on the category dummies.
- That is:
  - regress $Y_i = \text{FE}_1 + \text{FE}_2 + \epsilon_i$ and $x_i^w = \text{FE}_1 + \text{FE}_2 + \epsilon_i, \forall w$,
  - take residuals $\tilde{Y}_i = Y_i - \hat{Y}_i$ and $\tilde{x}_i^w = x_i^w - \hat{x}_i^w$
- Then use residuals as variables, in feature selection step or in machine learning task.

# 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 outputs an integer in some range (e.g. 1 to 10,000).

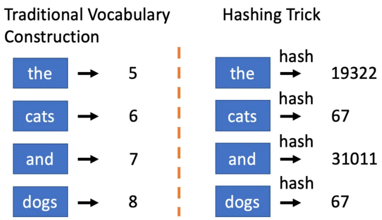
# Hashing Vectorizer




- 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 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



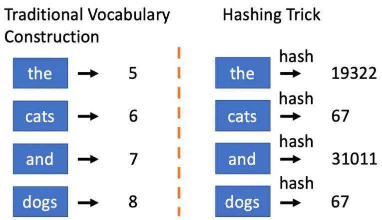

- Rather than make a one-to-one lookup for each n-gram, put n-grams through a hashing function that takes an arbitrary string and outputs an integer in some range (e.g. 1 to 10,000).

```
>>> 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.

# 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 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 – but the eli5 implementation keeps track of the mapping
- collisions – n-grams will randomly be paired with each other in the feature map.
  - usually innocuous, but could sum outputs of two hashing functions to minimize this.

# Collocations are Familiar N-grams

- 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")

# Collocations are Familiar N-grams

- 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")

# Collocations are Familiar N-grams

- 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 metric for identifying collocations is 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 $w_1\_w_2$.

# Point-wise mutual information

- A metric for identifying collocations is point-wise mutual information:

$$\text{PMI}(w_1, w_2) = \frac{\Pr(w_1\_w_2)}{\Pr(w_1)\Pr(w_2)}$$
$$= \frac{\text{Prob. of collocation, actual}}{\text{Prob. of collocation, if independent}}$$

  where $w_1$ and $w_2$ are words in the vocabulary, and $w_1, w_2$ is the N-gram $w_1\_w_2$.
  - ranks words by how often they collocate, relative to how often they occur apart.

# Point-wise mutual information

- A metric for identifying collocations is point-wise mutual information:

$$\text{PMI}(w_1, w_2) = \frac{\Pr(w_1\_w_2)}{\Pr(w_1)\Pr(w_2)}$$
$$= \frac{\text{Prob. of collocation, actual}}{\text{Prob. of collocation, if independent}}$$

  where $w_1$ and $w_2$ are words in the vocabulary, and $w_1, w_2$ is the N-gram $w_1\_w_2$.
  - ranks words by how often they collocate, relative to how often they occur apart.
- Warning: Rare words that appear together once or twice will have high PMI.
  - Address this with minimum frequency thresholds.

# Geometric Mean: Normalized PMI for $N \geq 2$

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

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

# Geometric Mean: Normalized PMI for $N \geq 2$

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

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

- E.g., for trigrams:

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

# Geometric Mean: Normalized PMI for $N \geq 2$

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

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

- 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.

# Phrase Dictionaries

- WordNet has some phrases as single entities.
- The Paraphrase Database 2.0 (PPDB, `paraphrase.org/#/download`) has a large database of equivalent/related words/phrases.
- Could take wikipedia article names as lists of multi-word expressions.
- In law, could use legal dictionaries (e.g., "first amendment", "beyond a reasonable doubt").

# Named Entity Recognition

- refers to the task of identifying named entities such as "ETH Zurich" and "Marie Curie", which can be used as tokens.

  [$_{\text{PER}}$ John Smith ] , president of [$_{\text{ORG}}$ McCormik Industries ] visited his niece [$_{\text{PER}}$ Paris ] in [$_{\text{LOC}}$ Milan ], reporters say .

# Named Entity Recognition

- refers to the task of identifying named entities such as "ETH Zurich" and "Marie Curie", which can be used as tokens.

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

works of art.

| Type | Tag | Sample Categories | Example sentences |
|---|---|---|---|
| People | PER | people, characters | **Turing** is a giant of computer science. |
| Organization | ORG | companies, sports teams | The **IPCC** warned about the cyclone. |
| Location | LOC | regions, mountains, seas | The **Mt. Sanitas** loop is in **Sunshine Canyon**. |
| Geo-Political Entity | GPE | countries, states, provinces | **Palo Alto** is raising the fees for parking. |
| Facility | FAC | bridges, buildings, airports | Consider the **Golden Gate Bridge**. |
| Vehicles | VEH | planes, trains, automobiles | It was a classic **Ford Falcon**. |

**Figure 18.1** A list of generic named entity types with the kinds of entities they refer to.

- Blackstone has a trained legal NER system in spaCy (for UK law).

# 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 parts of speech tags as features – e.g., using more adjectives, or using more passive verbs.

# 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 parts of speech tags as features – e.g., using more adjectives, or using more passive verbs.
- POS n-gam frequencies (e.g. NN, NV, VN, ...), like function words, are good stylistic features for authorship detection.
  - not biased by topics/content

# Constructing “Memes” with POS

- A: Adjective, N: Noun, V: Verb, P: Preposition, D: Determinant, C: Conjunction.
- 2-grams: AN, NN, VN, VV, NV, VP.
  - tax credit, magistrate judge
- 3-grams: NNN, AAN, ANN, NAN, NPN, VAN, VNN, AVN, VVN, VPN, ANV, NVV, VDN, VVV, NNV, VVP, VAV, VVN, NCN, VCV, ACA, PAN.
  - armed and dangerous, stating the obvious

# Constructing "Memes" with POS

- A: Adjective, N: Noun, V: Verb, P: Preposition, D: Determinant, C: Conjunction.
- 2-grams: AN, NN, VN, VV, NV, VP.
  - tax credit, magistrate judge
- 3-grams: NNN, AAN, ANN, NAN, NPN, VAN, VNN, AVN, VVN, VPN, ANV, NVV, VDN, VVV, NNV, VVP, VAV, VVN, NCN, VCV, ACA, PAN.
  - armed and dangerous, stating the obvious
- 4-grams: NCVN, ANNN, NNNN, NPNN, AANN, ANNN, ANPN, NNPN, NPAN, ACAN, NCNN, NNCN, ANCN, NCAN, PDAN, PNPN, VDNN, VDAN, VVDN.
  - `Beyond a reasonable doubt (preposition, article, adjective, noun)`
  - `Earned income tax credit (adjective, noun, noun, noun)`

# Ranking Partisan language

## Monroe et al (2009)

- This paper systematically explores a number of methods for identifying words that are distinctive of groups of speakers
  - in this case, whether U.S. congressmen are Republicans are Democrats.

# Ranking Partisan language

## Monroe et al (2009)

- This paper systematically explores a number of methods for identifying words that are distinctive of groups of speakers
  - in this case, whether U.S. congressmen are Republicans are Democrats.
- First, they separate speeches by topic using latent dirichlet allocation (next lecture).
  - they then test a number of methods for ranking partisanship of words.

# Relative Frequency of Words

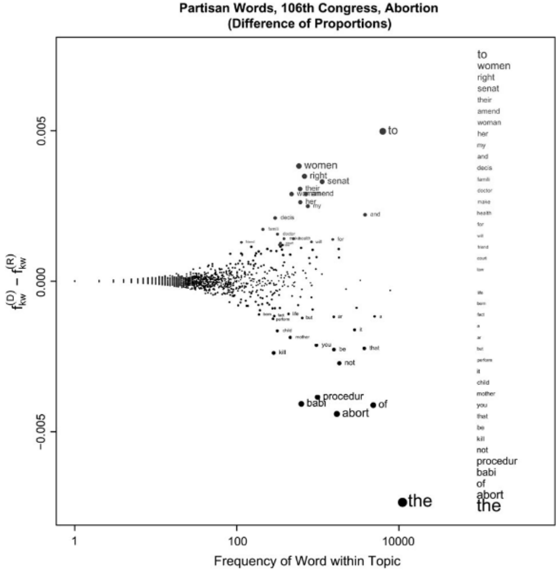



**Fig. 1** Feature evaluation and selection using $f_{kw}^{(D)} - f_{kw}^{(R)}$. Plot size is proportional to evaluation weight, $|f_{kw}^{(D)} - f_{kw}^{(R)}|$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right. The results are almost identical for two other measures discussed in the text: unlogged *tf.idf* and frequency-weighted WordScores.

# Log Odds Ratio Between Groups

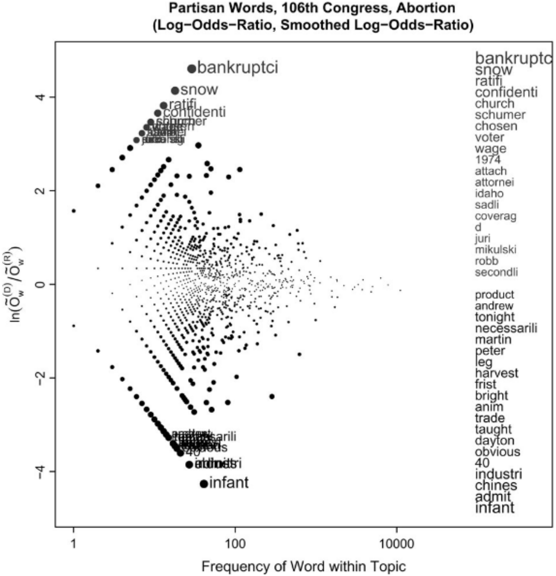



**Fig. 2** Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\left|\hat{\delta}_{kw}^{(D-R)}\right|$. Top 20 Democratic and Republican words are labeled and listed in rank order. The results are identical to another measure discussed in the text: the log-odds-ratio with uninformative Dirichlet prior.

# Bayesian Multinomial Model

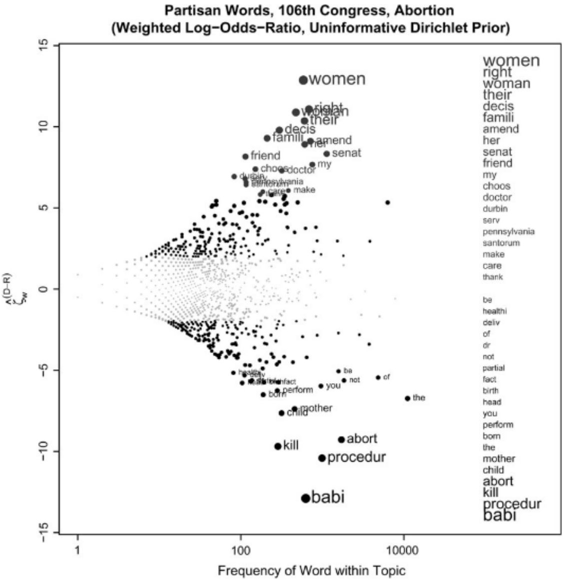



**Fig. 4** Feature evaluation and selection using $\hat{\zeta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\left|\hat{\zeta}_{kw}^{(D-R)}\right|$; those with $\left|\hat{\zeta}_{kw}^{(D-R)}\right|$<1.96 are gray. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

# Bayesian Multinomial Model, LaPlace Prior

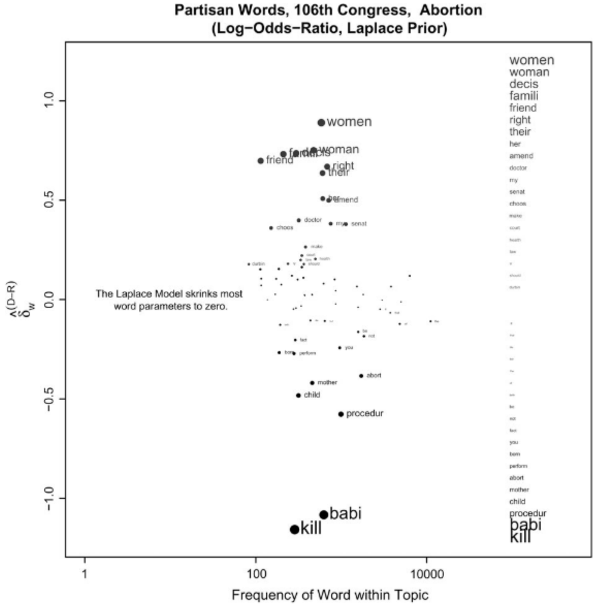



**Fig. 6** Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\hat{\delta}_{kw}^{(D-R)}$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

# Questions

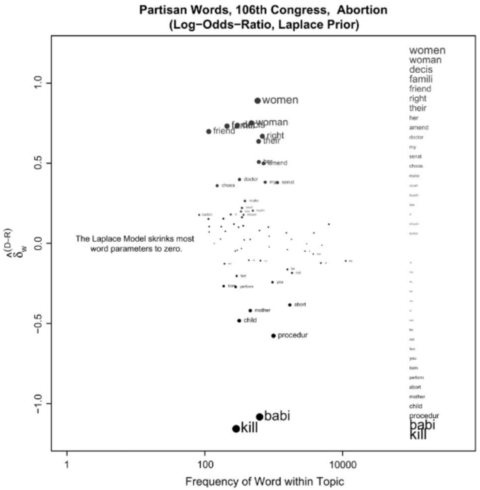



**Fig. 6** Feature evaluation and selection using $\hat{\delta}_{kw}^{(D-R)}$. Plot size is proportional to evaluation weight, $\hat{\delta}_{kw}^{(D-R)}$. The top 20 Democratic and Republican words are labeled and listed in rank order to the right.

- How robust across topics?
- Why not n-grams?
- Is this useful for anything besides description?
- Others?

# 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.

# 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.
- Deliverables:
  - description of topic (March 29th, 10% of grade)
  - proposal/outline (April 26th, 10% of grade)
  - Poster/presentation session (some time in May, 10% of grade).
  - Rough draft with data/methods/results (July 15th, 20% of grade)
  - Final draft (September 1st, 50% of grade)

# Previous Year's Projects (1)

# 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, one some VC funding

# 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, one some VC funding
- Another group partnered with a local company to build out environmental-regulation analytics
  - won an Innosuisse grant.

# 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, one some VC funding
- Another group partnered with a local company to build out environmental-regulation analytics
  - won an Innosuisse grant.

# Previous Year's Projects (2)

**Three projects have been published:**

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, and 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).

# Previous Year's Projects (2)

**Three projects have been published:**

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, and 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).

One more is under review at ACL:

- "MemSum: Extractive Summarization using Multi-step Episodic Markov Decision Processes" (Nianlong Gu, Elliott Ash, Richard Hahnloser).

# Previous Year's Projects (3)

A number of other projects that are likely to get published:

- "Kwame: A Bilingual AI Teaching Assistant for Online SuaCode Courses" (George Boateng 2020)
- a partisan tweet generator that responds in the style of a Republican or Democrat.
- an analysis of bias towards immigrants in the early 1900s using old newspapers.
- a causal analysis using deep instrumental variables of what arguments in judicial opinions increase citations
- a partisan question answering system that answers questions with a partisan slant.
- an audio analysis of european central bank speech recordings.

*Note: These projects were above expectation.*

# Questions / comments?

- As suggested, we will set up a meet-and-greet for those doing projects.