Natural Language Processing

Lecture 2

Spring 2023

RowanUniversity

Last Time

- Introduced the course, myself, yourselves
- Introduced NLP and the challenges in the field
- Talked methods for cleaning text, tokenization
- Used n-gram *language models* to capture statistics in text corpora

Today's Lecture

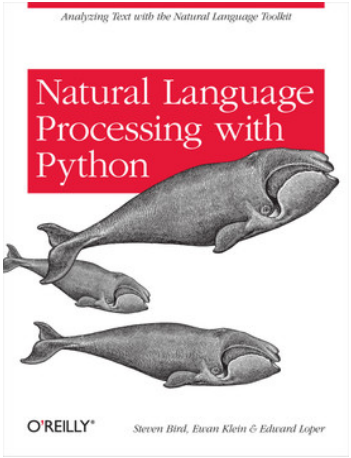


Figure: NLTK Textbook

What is NLTK?

- Light-weight python library for text processing
- Alternatives include gensim, spacy
- Developed at UPenn in 2001 for a course in computational linguistics

Modules

Language processing task	NLTK modules	Functionality
Accessing corpora	corpus	standardized interfaces to corpora and lexicons
String processing	tokenize, stem	tokenizers, sentence tokenizers, stemmers
Collocation discovery	collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	tag	n-gram, backoff, Brill, HMM, TnT
Machine learning	classify, cluster, tbl	decision tree, maximum entropy, naive Bayes, EM, k-means
Chunking	chunk	regular expression, n-gram, named-entity
Parsing	parse, ccg	chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	sem, inference	lambda calculus, first-order logic, model checking
Evaluation metrics	metrics	precision, recall, agreement coefficients
Probability and estimation	probability	frequency distributions, smoothed probability distributions
Applications	app, chat	graphical concordancer, parsers, WordNet browser, chatbots
Linguistic fieldwork	toolbox	manipulate data in SIL Toolbox format

Figure: Modules used in NLTK

Accessing data

- Recall: we call a collection of text used in NLP a *corpora*
- Modern NLP uses MASSIVE corpora (approx. 500B tokens)
- NLTK provides some smaller datasets to practice getting your feet wet
- NLTK also has a downloader function to get data
- `import nltk; nltk.download()`
- Saves to the python package filepath for ease of use

Text Documents

- The internal data structure for NLTK is a corpus, or sometimes `text`
- The number of tokens of a text can easily be fetched
- `len(text)`
- If the text is tokenized, tokens can be sorted
- `sorted(set(text))`
- Individual tokens can be counted and searched for
- `text.count('hello')`
- These counts can provide quick unigram probabilities
- $P('hello')$ =`text.count('hello') / len(set(text))`

Distributions

- NLTK also has fast, built-in functions for counting tokens (or other objects) from text
- `fdist = FreqDist(text)`
- `fdist.most_common(50)`
- The distribution is essentially a dictionary, so indices can be fetched
- `fdist['hello']`
- We can also get words with frequency 1, or *hapaxes*
- `fdist.hapaxes()`

More Counting

- We can also easily extract other counts, such as frequent bigrams
- `list(bigrams(text))`
- The above is a generator, aka `yield` will give an object rather than `return`
- Can also count the distribution of word lengths
- `FreqDist([len(x) for x in text])`
- These types of token, length, and other counts provide important statistics about a corpus that can be used in other ML processes
- For instance, word length used in ConvNets, sentence length used in RNNs

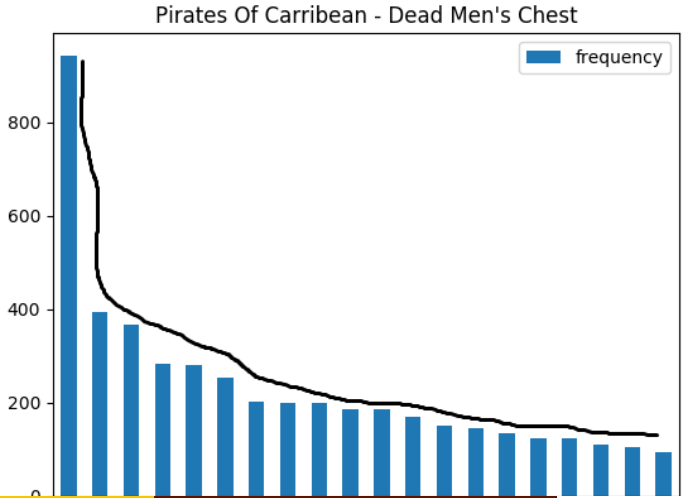
Zipf's Law

Definition

Zipf's law states that the frequency of any word in a corpus is inversely proportional to its rank in the frequency table.

Meaning, the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc...

Zipf's law


Pirates Of Carribean - Dead Men's Chest
frequency
800
600
400
200

Corpora and Lexical Resources

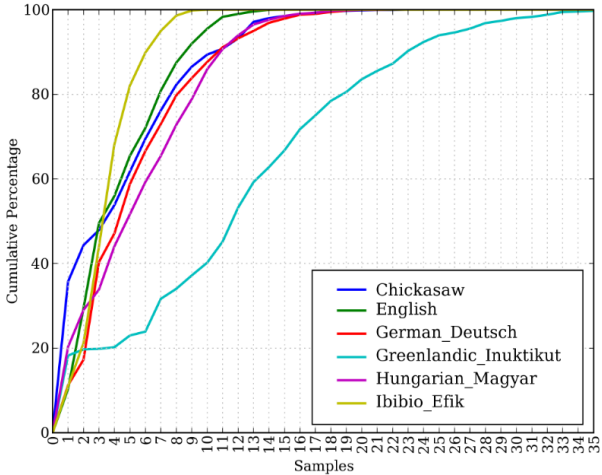
Common Corpora

- Project Gutenberg - 25k online books (we will use Shakespeare in Assignment 1)
- `nltk.corpus.gutenberg.fileids()`
- Web text, collection of 'noisier' works such as news articles and chats
- `from nltk.corpus import webtext`
- NPS - Naval Postgraduate School for internet predator detection
- `from nltk.corpus import nps_chat`
- The Brown corpus - 5M words categorized into genres
- `from nltk.corpus import brown; brown.categories()`
- The Reuters corpus - news articles split into train/test splits for prediction
- `from nltk.corpus import reuters; reuters.categories()`

Multi-lingual Corpora

- Universal Human Rights Declaration - 300 languages!
- `from nltk.corpus import udhr`
- “Unfortunately, for many languages, substantial corpora are not yet available.”
- This is a HUGE problem in NLP and computational linguistics!
- Many calls to action for under-resourced languages
- #BenderRule

Multi-lingual Analysis



Corpus Modules

Example	Description
`fileids()`	the files of the corpus
`fileids([categories])`	the files of the corpus corresponding to these categories
`categories()`	the categories of the corpus
`categories([fileids])`	the categories of the corpus corresponding to these files
`raw()`	the raw content of the corpus
`raw(fileids=[f1,f2,f3])`	the raw content of the specified files
`raw(categories=[c1,c2])`	the raw content of the specified categories
`words()`	the words of the whole corpus
`words(fileids=[f1,f2,f3])`	the words of the specified fileids
`words(categories=[c1,c2])`	the words of the specified categories
`sents()`	the sentences of the whole corpus
`sents(fileids=[f1,f2,f3])`	the sentences of the specified fileids
`sents(categories=[c1,c2])`	the sentences of the specified categories
`abspath(fileid)`	the location of the given file on disk
`encoding(fileid)`	the encoding of the file (if known)
`open(fileid)`	open a stream for reading the given corpus file
`root`	if the path to the root of locally installed corpus

Lexical Resources

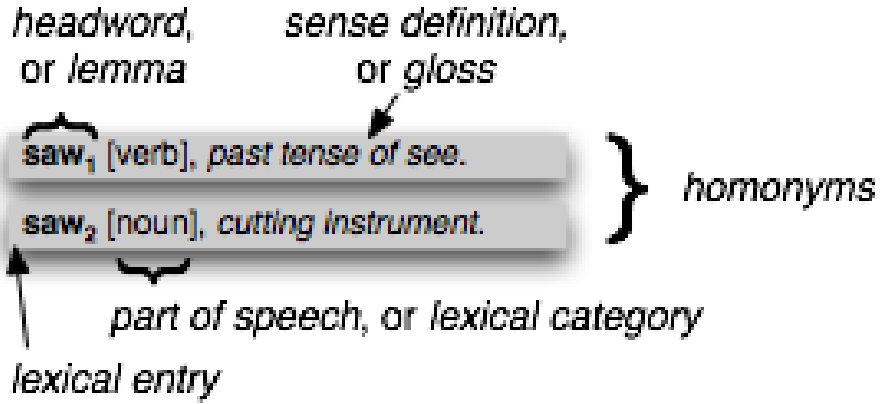
Definition

Lexicon A lexicon is the the complete set of meaningful units in a language.

Definition

Lemma A lemma is a theoretical abstract conceptual form of a word, representing a specific meaning, before the creation of a specific phonological form as the sounds of a lexeme, which may find representation in a specific written form as a dictionary or lexicographic word.

Lexicon


headword,
or lemma
sense definition,
or gloss
saw1 [verb], past tense of see.
saw2 [noun], cutting instrument.
homonyms
part of speech, or lexical category
lexical entry

Ambiguity

- Homonyms: distinct words with the same spelling, but different meanings
- Example: the ‘bank’ of a river vs. depositing money at the ‘bank’
- This type of ambiguity is what makes NLP hard!
- Whole field of word sense disambiguation
- Whole field of coreference resolution (i.e. ‘she’ went to the store)
- Whole field of long-term dependency tracking and syntactic parse trees
- Simple counts won’t help us in this area, we’ll need fancier tools...

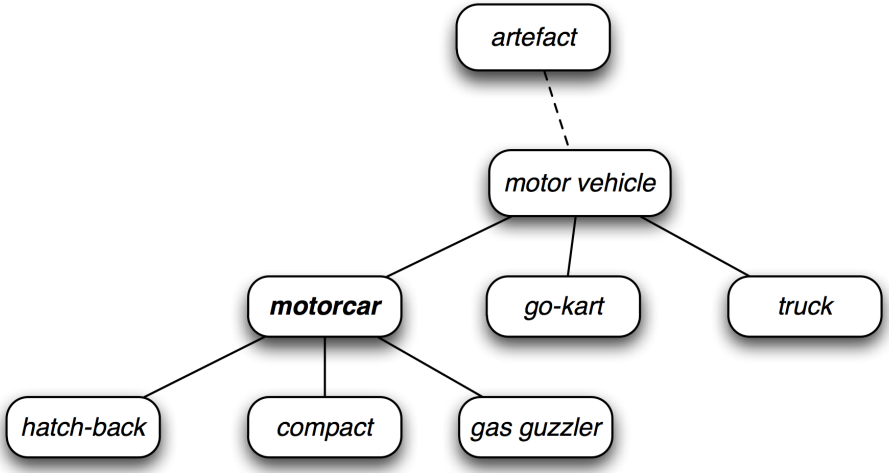
Other Lexical Tools

- List of English words `nltk.corpus.words.words()`
- Lists of stopwords `from nltk.corpus import stopwords; stopwords.words('english')`
- High frequency, low impact
- Names `nltk.corpus.names`
- So many other lists available, countries, company names, proper nouns, etc.
- Lists used in NLP are typically called gazetteers

WordNet

- Potentially the most valuable lexical resource for English
- Semantics is the study of meaning in language (or data)
- WordNet focuses on the *semantic* structure of words, i.e. how they relate and interact with one another
- The main structure in WordNet is the `synset`, of set of synonyms
- `from nltk.corpus import wordnet as wn`
- `wn.synsets('motorcar')`
- `wn.synset('car.n.01').lemma_names()`
- `wn.synsets('car')`
- `for synset in wn.synsets('car'): print(synset.lemma_names())`

WordNet Hierarchy



Navigating WordNet

- WordNet is a graph whose edges can be navigated through hypernym/hyponym relations
- i.e. the relation between superordinate and subordinate concepts
- `motorcar = wn.synset('car.n.01'); types_of_motorcar = motorcar.hyponyms()`
- `motorcar.root_hypernyms()`
- `nltk.app.wordnet()`

More Navigation

- Other relations (or predicates) exist in WordNet as well
- Relate items to their components (meronyms)
- Relate items to the things they are contained in (holonyms)
- Relationships between verbs (entailments)
- `wn.synset('walk.v.01').entailments()`

Assessing Similarity

- As WordNet is a graph, we can use the path length as a feature to assess the similarity between two concepts
- This similarity is on a $[0-1]$ range, higher, more similar
- `tortoise = wn.synset('tortoise.n.01')`
- `orca = wn.synset('orca.n.01')`
- `novel = wn.synset('novel.n.01')`
- `orca.path_similarity(tortoise)`
- `novel.path_similarity(tortoise)`
- We'll return to this example in two weeks, similarity is a critical component of NLP!

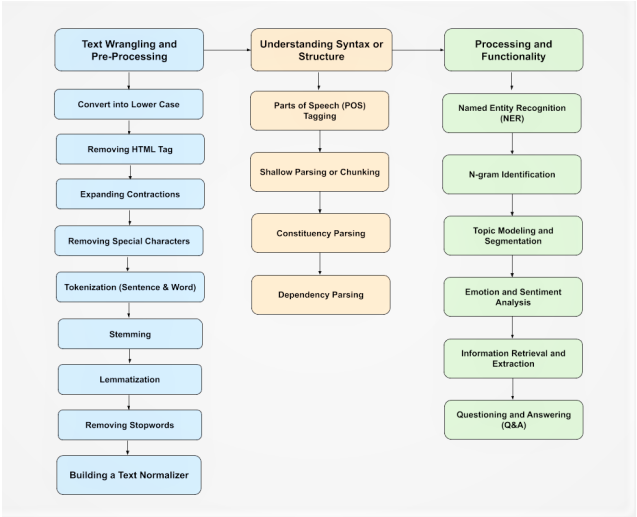
Taking Stock

So far, we have covered...

- n-gram language models
- accessing lexical resources
- ... great, but what do we do with these tools?
- Next: pre-processing of text for downstream use
- Last time, we saw tokenization and regular expressions, this time, a little more

Pre-processing pipeline

Pipeline


Text Wrangling and Pre-Processing
Convert into Lower Case
Removing HTML Tag
Expanding Contractions
Removing Special Characters
Tokenization (Sentence & Word)
Stemming
Lemmatization
Removing Stopwords
Building a Text Normalizer
Understanding Syntax or Structure
Parts of Speech (POS) Tagging
Shallow Parsing or Chunking
Constituency Parsing
Dependency Parsing
Processing and Functionality
Named Entity Recognition (NER)
N-gram Identification
Topic Modeling and Segmentation
Emotion and Sentiment Analysis
Information Retrieval and Extraction
Questioning and Answering (Q&A)

Tokenizers (Revisited)

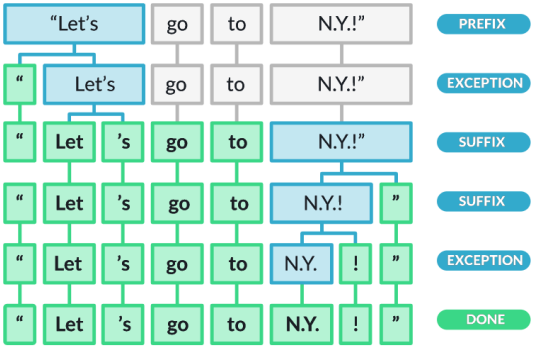
- There are many ways to tokenize a sentence
- `Good muffins cost $3.88n in New York. Please buy me... two of them.nn Thanks.`
- Easiest: White-space `from nltk.tokenize import word_tokenize`
- Next: include punctuations as well `from nltk.tokenize import wordpunct_tokenize`
- Problem: doesn't respect sentence boundaries
- Soln. use the sentence tokenizer first! `from nltk.tokenize import sent_tokenize, word_tokenize`
- Other relevant tokenizers: regex `from nltk.tokenize import RegexpTokenizer`
- These tokenizers are destructive: can't recover the original string!

Non-destructive Tokenization

- We need some way of keeping track of where token splits come from to rebuild text
- One such way is to use a **span** tokenizer
- `from nltk.tokenize import WhitespaceTokenizer; list(WhitespaceTokenizer().span_tokenize(s))`
- These are not natively supported in NLTK `Text`
- These ARE supported when using tooks like SpaCy
- Compared to NLTK, the natural data structure in SpaCy is a `Doc`
- We talked about BPE for tokenization last time, more on that later

SpaCy

If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings. This way, spaCy can split **complex, nested tokens** like combinations of abbreviations and multiple punctuation marks.



Normalization

- Once we have out text converted to tokens, we need to figure out what to do with them
- Looking for a *normalized* form of each token
- Does case matter? 'Drive' vs. 'drive' vs. 'DRIVE'
- `words = [w.lower() for w in tokens]`
- Are there special characters? Emojis? Encoding?
- `string.encode('utf8')`
- How much **semantic** information does 'drive' have vs. 'drives'?
- Not much, common endings can be removed in a process called **stemming**

Stemming

- Could define common stems and use regex tokenizer
- `for suffix in ['ing', 'ly', 'ed', 'ious', 'ies', 'ive', 'es', 's', 'ment']`
- Porter's stemming algorithm
- `porter = nltk.PorterStemmer()`
- Lancaster's stemming algorithm
- `lancaster = nltk.LancasterStemmer()`
- "Stemming is not a well-defined process, and we typically pick the stemmer that best suits the application we have in mind"

Porter's Stemming Algorithm

Porter Stemmer

Based on a series of rewrite rules run in series

- A cascade, in which output of each pass fed to next pass

Some sample rules:

$$\begin{aligned} \text{ATIONAL} &\rightarrow \text{ATE} \quad \text{(e.g., relational} \rightarrow \text{relate)} \\ \text{ING} &\rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\ \text{SSES} &\rightarrow \text{SS} \quad \text{(e.g., grasses} \rightarrow \text{grass)} \end{aligned}$$

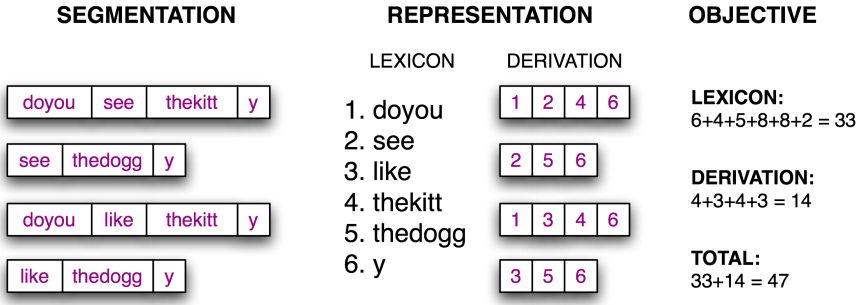
Lemmatization

- Converts foken *forms* to their base concept
- For instance, ‘women’ converts to ‘woman’ (singular form)
- `wnl = nltk.WordNetLemmatizer()`
- Operates by trying to strip affixes and comparing to a dictionary of known terms
- Due to lookups, lemmatization can be slow
- We will see that modern pipelines are less reliant on lemmas

Segmentation

- For some languages or domains, tokenization is much harder
- Example: Chinese
- (ai4 "love" (verb), guo2 "country", ren2 "person") could be tokenized as / , "country-loving person" or as / , "love country-person."
- Example: DNA sequences aggggaattcaggagacggtcattccatcgcccaagcaacatggcccgcacccctgg
- Example: SMILES formulas CC(CCC(=O)N)CN

Segmentation



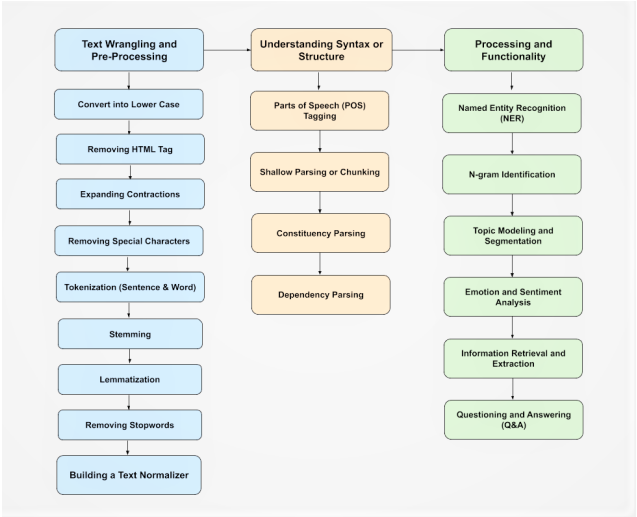
Simulated Annealing

```
from random import randint

def flip(segs, pos):
    return segs[:pos] + str(1-int(segs[pos])) + segs[pos+1:]

def flip_n(segs, n):
    for i in range(n):
        segs = flip(segs, randint(0, len(segs)-1))
    return segs

def anneal(text, segs, iterations, cooling_rate):
    temperature = float(len(segs))
    while temperature > 0.5:
        best_segs, best = segs, evaluate(text, segs)
        for i in range(iterations):
            guess = flip_n(segs, round(temperature))
            score = evaluate(text, guess)
            if score < best:
                best, best_segs = score, guess
        score, segs = best, best_segs
        temperature = temperature / cooling_rate
        print(evaluate(text, segs), segment(text, segs))
    print()
    return segs
```

Stopword Removal

- Some utility words have little *semantic* impact on the text
- so.... remove them!
- `from nltk.corpus import stopwords; sw = stopwords.words('english')`
- `[x for x in sent if x not in sw]`

Pipeline (again)



Normalization

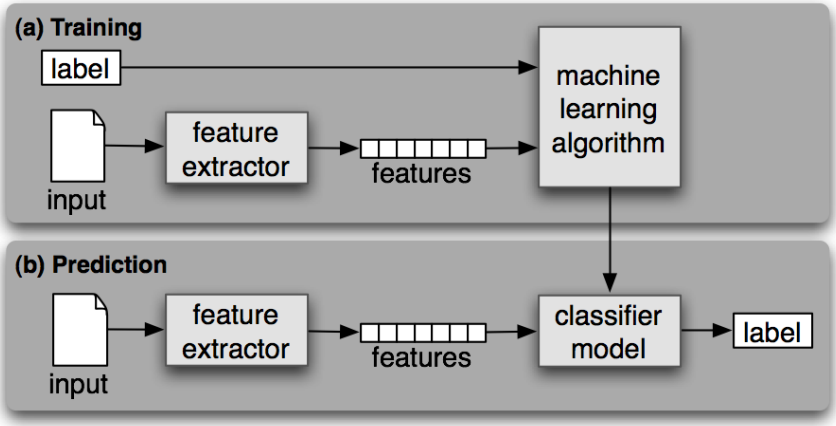
- Still tons of other pre-processing to cover: part-of-speech tagging, named entity recognition, dependency parsing.... see Chapter 5 for details if interested
- NLTK methods for the above aren't the best, Stanford and SpaCy have better software we will use later on
- With all the steps covered, they can be compiled into the final normalization process
- See this script as an example of a normalizer

Break

Features

- Now that we have pre-processed our text, what do we do with it?
- Machine learning methods (classification or regression) need input **features** in order to operate
- Features are typically numerical, but we still have text
- Need: some methods to convert text to features

Classification



ML Tasks

- Binary classification: email is spam or not
- Multi-class: sentiment is positive, negative, neutral
- Regression: given a grant proposal, how much funding should be given
- All require us to build robust features that will generalize well to unseen instances
- Let's look at the name classification example

Better Features

- Obviously, using the last letter isn't a great feature
- Could hand build other binary or numeric features, such as 'contains cat', 'is proceeded by the', 'had an upper case character'
- This would be time consuming, but is how NLP waas done for a very long time
- Advent of other **feature engineering pipelines** made this much easier

Engineered Features

- Word count matrices
- Let the column space be identifiers for all the documents
- Let the row space be all the tokens in the vocabulary
- Example:
  - S_1 = 'the cat in the hat'
  - S_2 = 'my cat ate my hat'
  - already note: two somewhat different uses of 'cat'
- V = `['the', 'cat', 'in', 'ate', 'my', 'hat']`

Vectorization

- This specific strategy (tokenization, counting and normalization) is called the Bag of Words (BOW) approach
- S_1 = 'the cat in the hat'
- S_2 = 'my cat ate my hat'
- V = `['the', 'cat', 'in', 'ate', 'my', 'hat']`

doc	the	cat	in	ate	my	hat
S_1	2	1	1	0	0	1
S_2	1	1	0	1	1	1

- The resulting feature matrix is called the term-frequency matrix (TF)

Dealing with Sparsity

- If the vocabulary size is very large, most of the TF entries will be zero
- This tends to cause computational issues, overflow errors
- Scipy sparse matrices can be used to limit this issue
- Another idea is to set a threshold for inclusion, i.e. if a word appears less than N times in the corpus, don't use it as a vocabulary feature
- `from sklearn.feature_extraction.text import CountVectorizer`
- Optional controls include minimum document frequency, maximum document frequency, maximum number of features, many others as well

TF Demo

Re-weighting Schemes

- Some tokens will occur VERY frequently in every document
- Example: looking at emails 'Dear soandso'
- Need a way to reduce the 'feature inflation' caused by those counts, if those features appear too important, the resulting classifier will fail
- Look to the frequency across all documents, re-weight by the inverse: **inverse document frequency**
- Now we build two matrices, TF and IDF, and compute their matrix product
- Issues with dividing by zero, so add plus one smoothing (like we saw last time, too)
- `from sklearn.feature_extraction.text import TfidfVectorizer`

TF x IDF Demo

1. For really large vocabularies, this can be a bottleneck still
2. Issues with generalization: seeing a new token may not have a corresponding feature
3. No word ordering is used, a very important part of human grammar!
4. **Next time: better features using neural networks**

Questions?