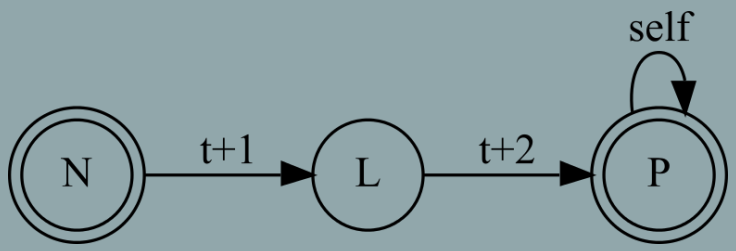
LING-362

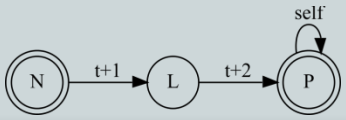
Introduction to **N**atural **L**anguage **P**rocessing

Topic Modelling II



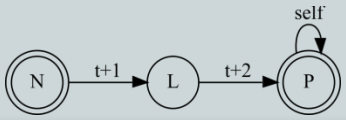
Topic Modelling

- From form to meaning!
- Today:
  - More on documents as data points
  - TF-IDF
  - Latent Dirichlet Allocation (LDA)



Basic MWE merging – *multiword.py*

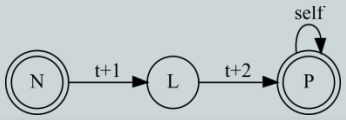
```
from nltk.tokenize import MWETokenizer
from nltk import FreqDist

mwe_list = [('palm', 'fronds'), ('coast', 'guard')]
mwe = MWETokenizer(mwe_list, separator='_')

tokens = word_tokenize("The coast guard saw palm fronds.")

mwe_tokenized = mwe.tokenize(tokens)

for term, freq in FreqDist(mwe_tokenized).most_common(10):
   print(term + "\t" + str(freq))
```



Pretrained models

- General NLP pipelines: Spacy, CoreNLP, gensim…
- Or try Nathan Schneider's MWE system:
  - https://github.com/nschneid/pysupersensetagger
- Trained on STREUSLE corpus:
  - https://github.com/nert-nlp/streusle

- You can also try writing your own
  (hint: this can be formulated as a BIO sequence labeling task)

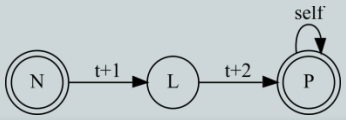


Collapsed frequencies: ‘palm fronds’

- Collapsed frequencies:
  - island 3
  - coast_guard 2
  - navy 1
  - u.s._navy 1
  - palm_frond 1
  - castaway 1
  - Honolulu 1
  - Fanadik 1
  - strand 1

Should we lower case?

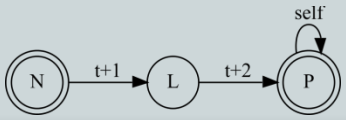
Are MWEs sometimes harmful?

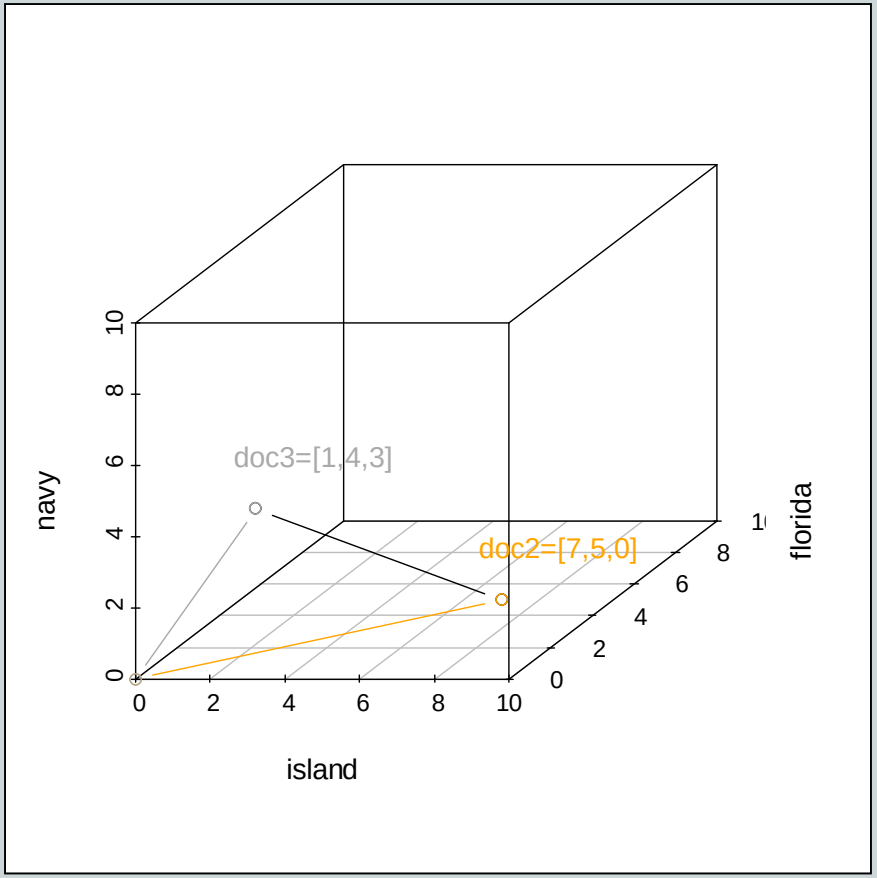


How to measure distance?

- We can measure distance in *n* dimensions
- Problems:
  - **Document length**
  - **Scaling**
  - **Term specificity**
  - **Collinearity**

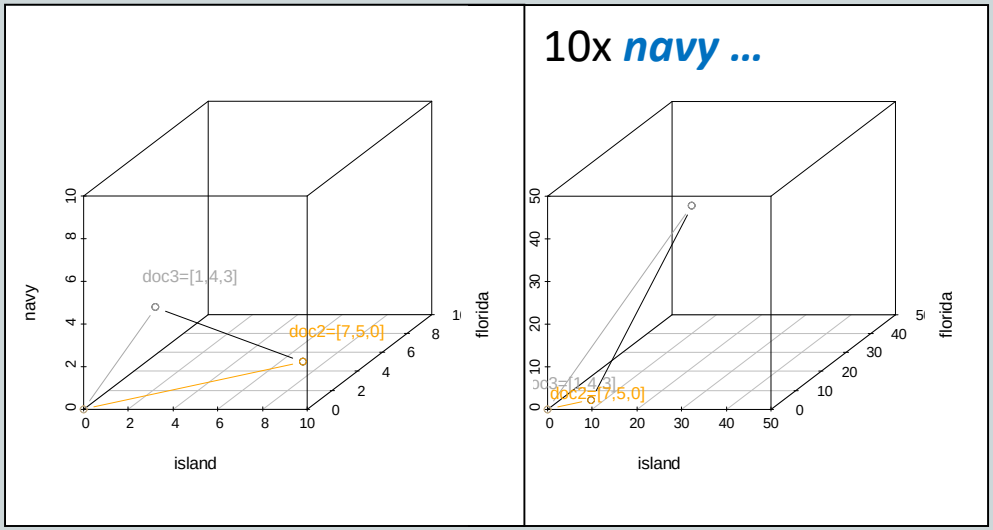
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

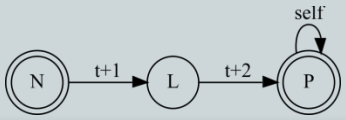




Document length

- Distance might seem like a good idea but…
  - Longer documents have more words
    - Short document may not mention U.S. Coast Guard often
    - But the fact that it does so in just 100 words seems significant
  - Still, **the angle** remains the same
  - ➢ Normalize length
  - ➢ **Cosine similarity**



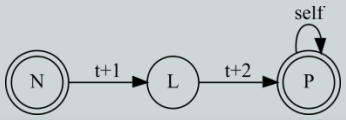


Cosine similarity

- Measure the **angle** between vectors:

$$\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

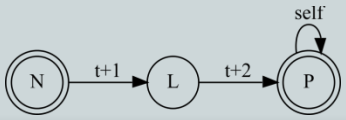
  - Dot product of two vectors divided by the product of their magnitudes
    - Dot product: multiply vectors cell-wise and sum
    - Magnitude: $\|X\| = \sqrt{x_1^2 + x_2^2 .. + x_n^2}$



Scaling

- With cosine similarity, the proportion of word frequencies gives the direction
  - If word 1 appears **once** and word 2 appear **twice:**
    - proportion 1:2
  - Now consider words appearing 10 vs. 20 times:
    - proportion 1:2

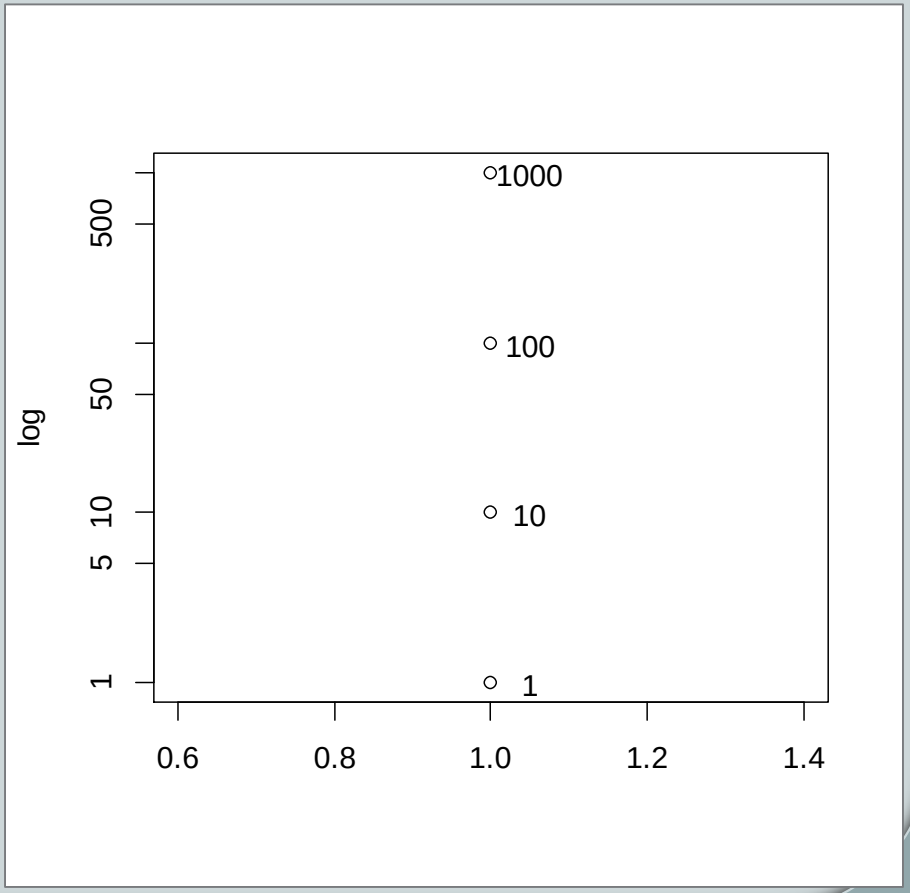
- But is a word appearing twice really twice as important as one appearing once?
- Is it the same for 10 vs. 20?

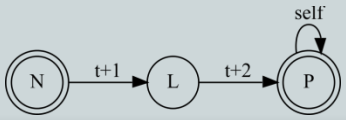


Scaling

- A common solution is to take **log** frequencies
  - E.g. log base 10
  - Difference between 1 and 10 same as difference between 10 and 100, 100 and 1000, …

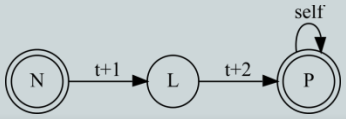
```
from math import log10
print(log10(5))
```





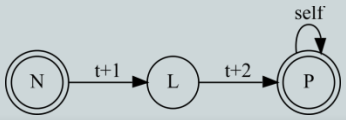
Term specificity

- A more fundamental problem with VSMs is that we have different ideas about what's important
  - Very frequent (non-stop) word is important?
  - Suppose our document contains:
    - ***insurance*** 10
    - ***try*** 10
    - ***story*** 10
  - What is it about?
  - How can we tell which is more important?



Collection frequency

- Some terms might generally be very frequent
  - Appearance less surprising – assign less importance
  - Use **Collection Frequencies** (sum over all documents, adapted NYT example from Manning & Schütze 1999)
    - ***insurance*** 10440
    - ***try*** 10422
    - ***story*** 23591 (less surprising)
  - How are ***insurance*** and ***try*** still different?

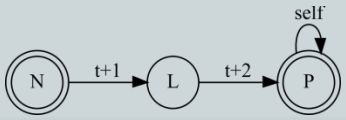


Document frequency

- Even with equal collection frequency, terms appear in different amounts of **documents**

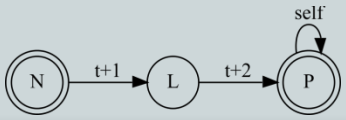
	term	*collection*	*document*
• ***insurance***	10	10440	3997
• ***try***	10	10422	8760
• ***story***	10	23591	10897

Can we combine these somehow?



Just one number

- To get just one number representing a term's relevance in a document:
  - Use log term frequency (TF): log(TF)
  - Weight it by proportion of documents with this term (DF) in an N document collection
- But we want **inverse** weighting – high document count is bad, so:
  - Inverse document frequency (IDF): log(N/DF)



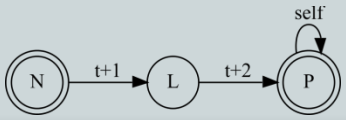
TF-IDF

- Most common weight function in Information Retrieval:
  - Weight for term *i* in document *j* (or 0 if unattested):

$$\text{weight}(i,j) = \left(1 + \log\left(TF_{i,j}\right)\right) \cdot \log\left(\frac{N}{DF_i}\right)$$

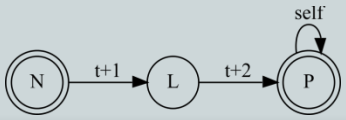
  - The IDF weighting for a unique term is maximal: log(N)
  - For a term appearing in all documents: log(1) = 0

- TF-IDF weights can be applied to frequencies in a BOW model



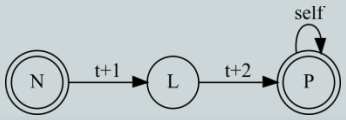
Classifying documents

- TF-IDF is great for finding distinctive terms
- But it doesn't tell us the best way to segment a collection into topics
  - We want to identify the most different kinds of documents
  - Words that characterize these 'kinds'
  - Degree of belonging to each of n topics, for each document (multiple topics possible)



An approach to automatic “topics”

- Latent Dirichlet Allocation (LDA) assumes:
  - Words can have their own prior probabilities in each possible topic
  - Assume that any set of documents we see is an example of the topic-based probabilities to realize each word
  - Each document is a mixture of the topics that generated it

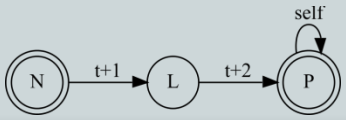


LDA – a caricature

- Suppose we have 10 topics with different probabilities for the same words:
  - Mary cooked up a new schematic
- Probably these words were generated according to these topics:
  - Mary cooked up a new schematic

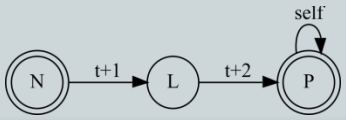
P(Mary|religion) > P(Mary|cooking) > …

Legend:
Religion
Cooking
Engineering
…

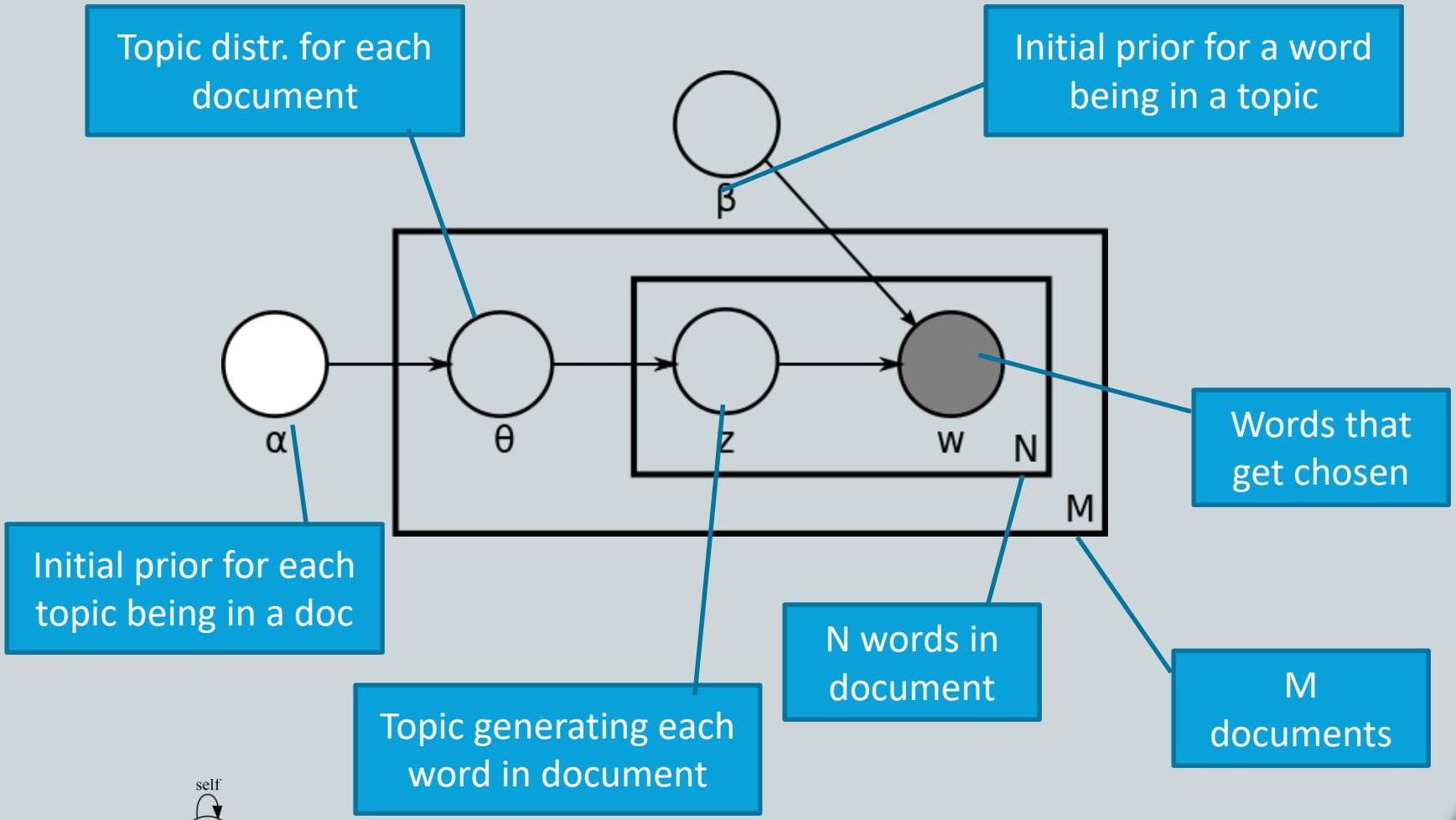


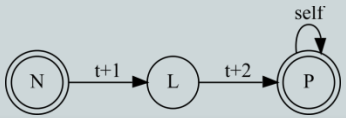
The 'generative story'

- According to LDA, documents are born like this
  - For every document, some random mix of topics is selected: 20% politics, 31% religion …
  - Once those are known, each position in the document is generated by some topic: randomly, word 1 gets to come from the 'religion' topic
  - Now a specific word is picked at random, based on its probability in that topic – very likely to be 'church', unlikely to be 'pizza' (but possible)



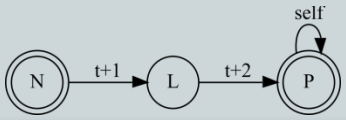
LDA – more formally





Inferring latent variables

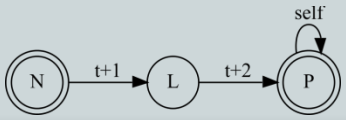
- Now the problem:
  - given the words, some idea of how many topics we might have and what prior distributions are like (incl. likelihood of each word)…
  - Infer the latent variables that generated each document
- Specifically – we want ϑi for each document i
  - Because if we know what words come from which topic with what likelihood…
  - We can get the topic mixture that generated that document with the highest likelihood



Let's do it!

- There are several methods to infer ϑi
  - Often: Gibbs sampling (similar to MCMC)
  - Gamble on the parameters, see if you get something like our collection, if not change parameters
  - Initially assume each word comes from a random topic – get ϑ, α and β
  - Run through data again – is this result likely? -> change
- We can't get into these methods in depth in this course
- But we can use some libraries to do this for us

- Further reading: Blei et al. (2003), Grus (2015)

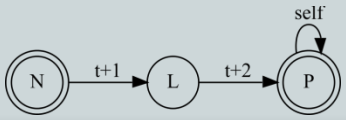


Library lda

- First we install the lda library from the command line:

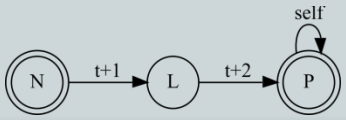
  > pip install lda

- Code in ***lda_example.py***



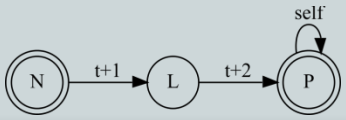
Imports

```
# Example adapted from Chris Strelioff
from numpy import argsort
import lda.datasets
```



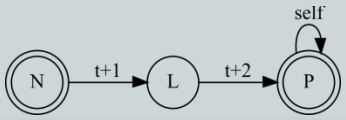
Example data

```
# Get some actual document data
# This is a two dimensional table of
# documents in each row, word frequencies in each column
reuters_data = lda.datasets.load_reuters()

# Get a list of document titles to help interpret results –
# corresponds to each row in the table
reuters_titles = lda.datasets.load_reuters_titles()

# Get the vocabulary in the documents - corresponds to each
# column in the table
reuters_vocab = lda.datasets.load_reuters_vocab()
```



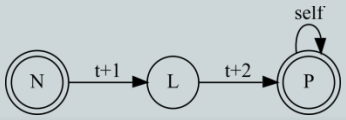
Testing the data

```
print("We are classifying " + str(len(reuters_titles)) + " documents")
print("with " + str(len(reuters_vocab)) + " distinct words.")

-- We are classifying 395 documents
-- with 4258 distinct words.

print("For example the title of document 5 is: " + reuters_titles[5])

-- For example the title of document 5 is:
-- 5 INDIA: Mother Teresa's condition unchanged, thousands pray. CALCUTTA

print("Word 4 is: " + reuters_vocab[4])
print("Its frequency in document 5 is: " + str(reuters_data[5][4]))
-- Word 4 is: mother Its frequency in document 5 is: 24
```



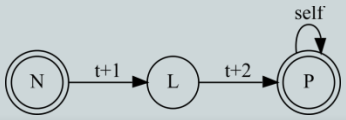
Fitting the model

```
lda_model = lda.LDA(n_topics=20, n_iter=500)
lda_model.fit(reuters_data)

topic_word_mapping = lda_model.topic_word_

# Let's check the probability of 'mother' (word 4) in topic 3
# Notice that numpy n-dimensional arrays use
# commas between dimensions (like R)
print("The probability of word 4 in topic 3 is:")
print(topic_word_mapping[3,4])

-- The probability of word 4 in topic 3 is:
-- 2.70009018301e-06
```



Getting top words for each topic

```
# Checking the top words for each topic:
print("\nThe top 5 words in each topic:\n" + "="*50)

for topic in topic_word_mapping:
    # Get list of words for this topic from the mapping,
    # sorted by descending probability
    words_in_topic = []
    sorted_indices = list(argsort(topic))[::-1]
    for i in range(5):  # Get top 5
        index = sorted_indices[i]
        words_in_topic.append(reuters_vocab[index])
    print(", ".join(words_in_topic))
```



Output

The top 5 words in each topic:

==========

world, million, against, group, court

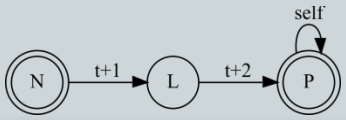
harriman, clinton, u.s, ambassador, paris

pope, vatican, surgery, hospital, rome

died, king, service, funeral, michael

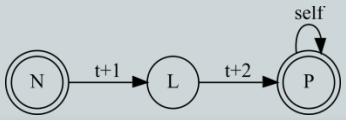
russian, russia, soviet, moscow, communist

…



Getting the best topic per document

```
# Check the top topic for each document
doc_topic_mapping = lda_model.doc_topic_

# Let's see if the first 10 cluster nicely
for n in range(10):
    # argmax returns the column with the maxmimum value for this row
    best_topic = doc_topic_mapping[n].argmax()
    print("doc" + str(n) + ", titled: " + reuters_titles[n])
    print("Best topic: " + best_topic)
```



Output

doc0, titled: 0 UK: Prince Charles spearheads British royal revolution. LONDON 1996-08-20
Best topic: 10

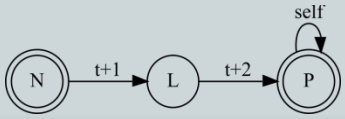
doc1, titled: 1 GERMANY: Historic Dresden church rising from WW2 ashes. DRESDEN, Germany 1996-08-21
Best topic: 4

doc2, titled: 2 INDIA: Mother Teresa's condition said still unstable. CALCUTTA 1996-08-23
Best topic: 15

doc3, titled: 3 UK: Palace warns British weekly over Charles pictures. LONDON 1996-08-25
Best topic: 10

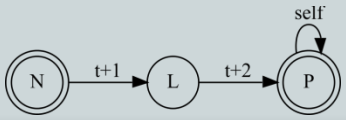
doc4, titled: 4 INDIA: Mother Teresa, slightly stronger, blesses nuns. CALCUTTA 1996-08-25
Best topic: 15

doc5, titled: 5 INDIA: Mother Teresa's condition unchanged, thousands pray. CALCUTTA 1996-08-
Best topic: 15

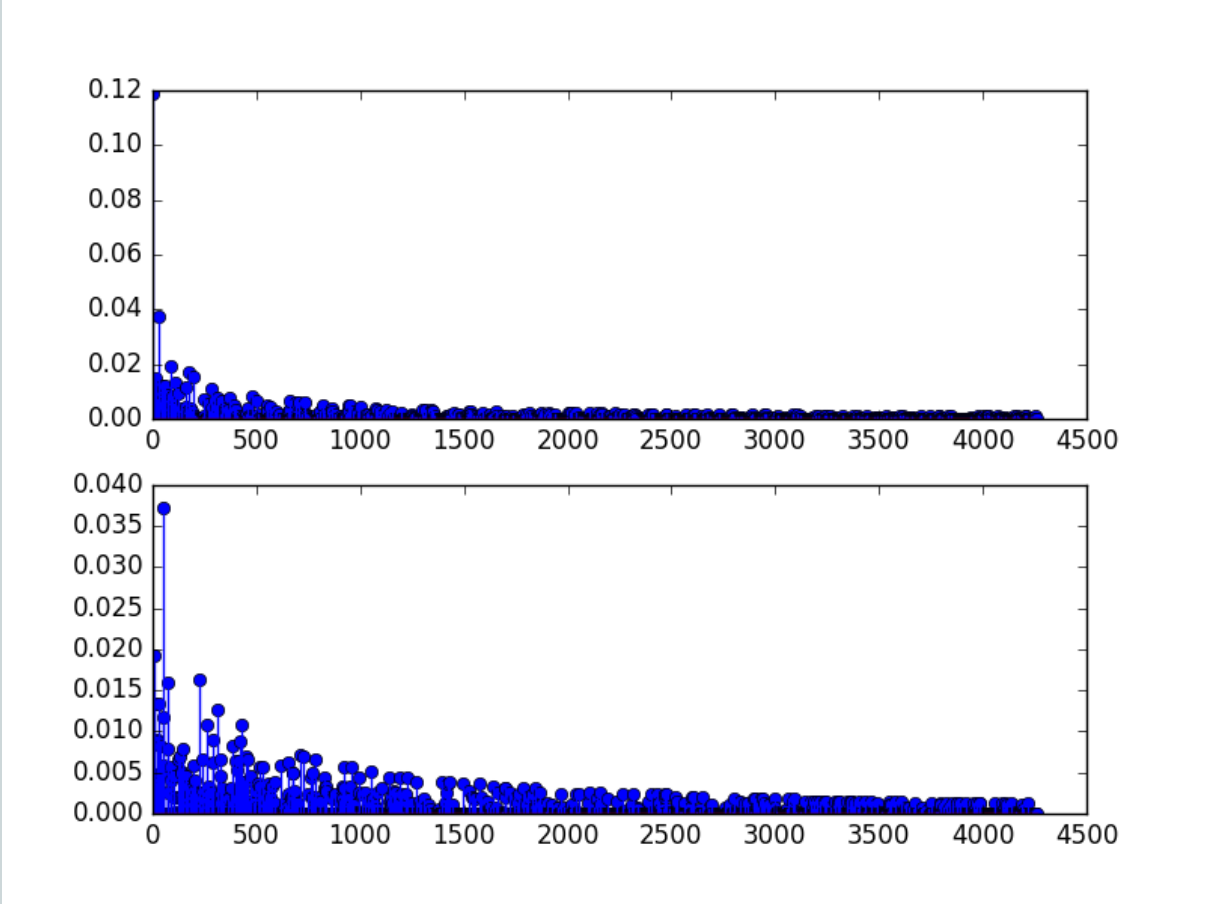


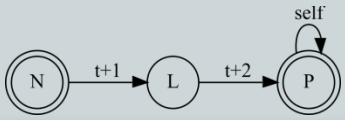
Plotting word distributions

```
import matplotlib.pyplot as pplt

# Make two rows, one column of plots
figure_container, my_plot_axes = pplt.subplots(2, 1)

# Fill the subplots with each of the following two topics
my_plot_axes[0].stem(topic_word_mapping[4,:])
my_plot_axes[1].stem(topic_word_mapping[15,:])

pplt.show()
```



Reminder: all words are in all topics!





What else can we read?

- If you want more practice, work through the NLTK book, chapter 8, up to section 5
  - Review of constituent parsing
  - Some additional ideas about sentence structure

- What to read next?
  - After the final: I recommend Chapter 6 – supervised text classification with some more advanced Python