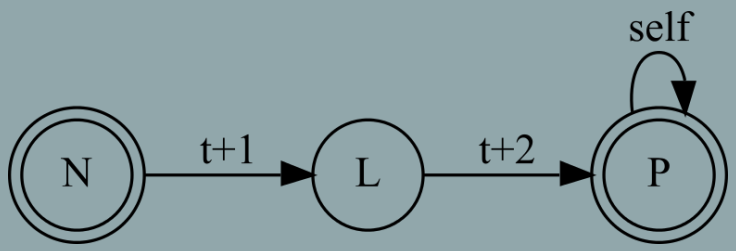
LING-362

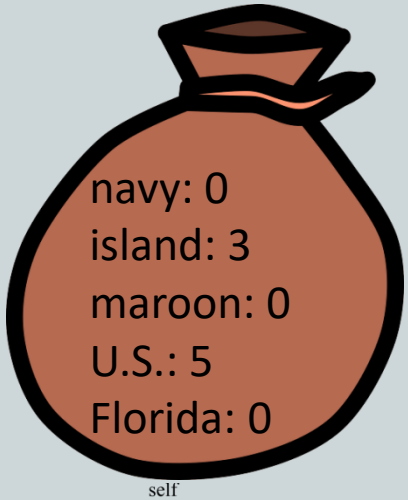
# Introduction to **N**atural **L**anguage **P**rocessing

Topic Modelling III

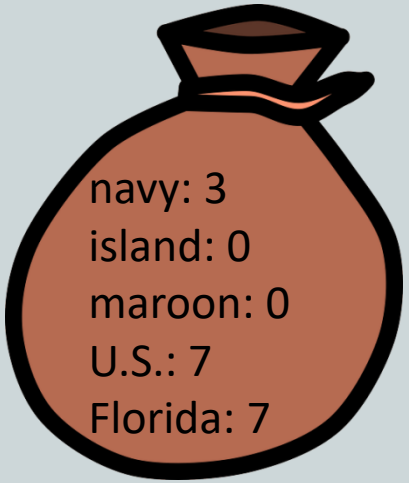


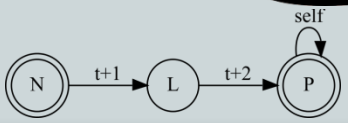
# The bag of words model - review

- Documents as frequency lists
- No information about **word order**
- Sparse data: lots of 0's



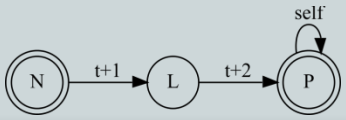




# Bags of words as vectors in a matrix

| | doc1 | doc2 | doc3 |
|---|---|---|---|
| navy | 0 | 3 | 3 |
| Island | 3 | 2 | 0 |
| maroon | 0 | 4 | 0 |
| U.S. | 5 | 1 | 7 |
| Florida | 0 | 1 | 7 |

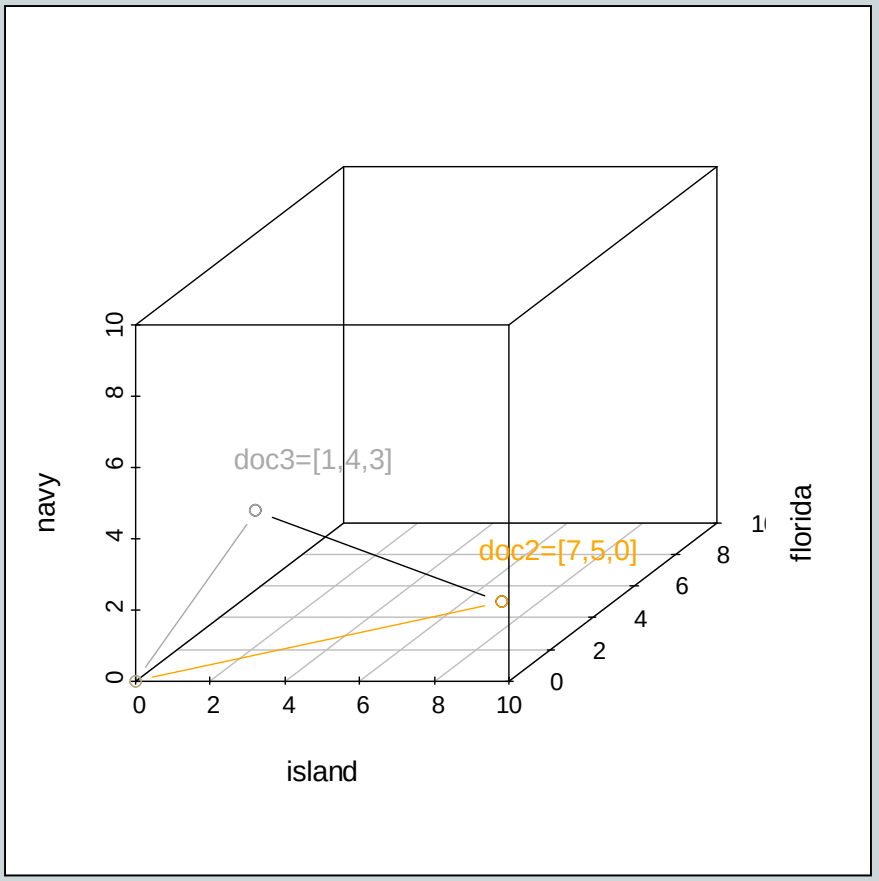


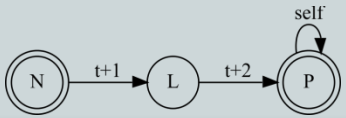
# How to measure distance?

- Documents as vectors
- Similarity as distance

$$\sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}$$

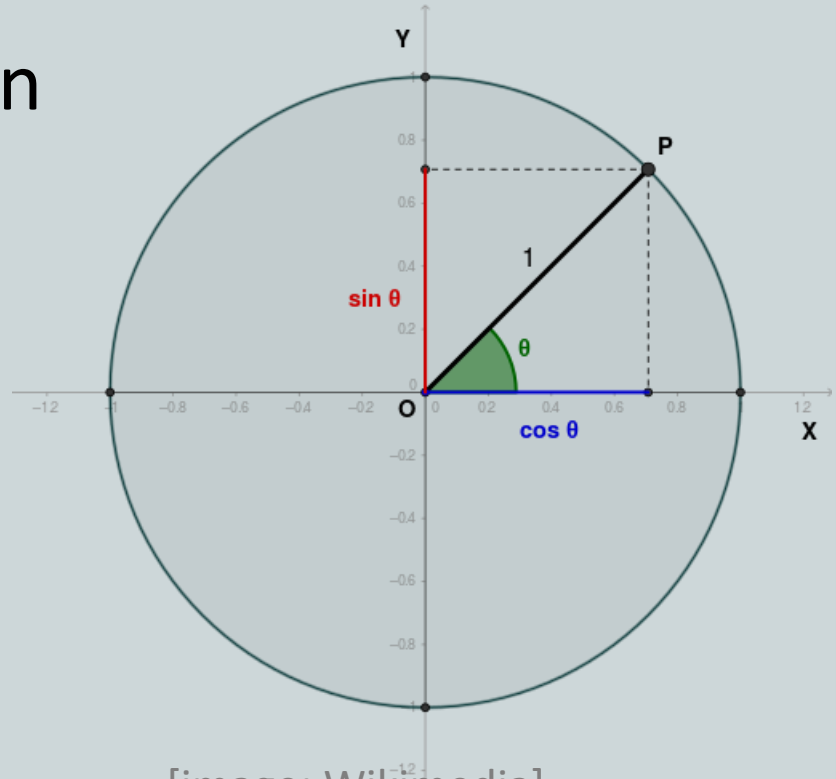
- Problems:
  - **Document length**
  - **Scaling**
  - **Term specificity**
  - **Collinearity**



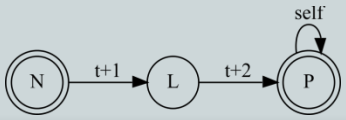



# Cosine similarity

- Works much like cosine in trigonometry:
  - **1 -> zero angle (same)**
  - **0 -> orthogonal (90°)**
  - **-1 -> opposite (180°)**




[image: Wikimedia]

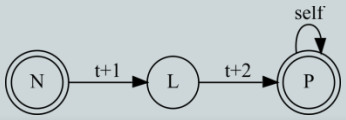


# Cosine similarity

- Measure **angle** between vectors for similarity:

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

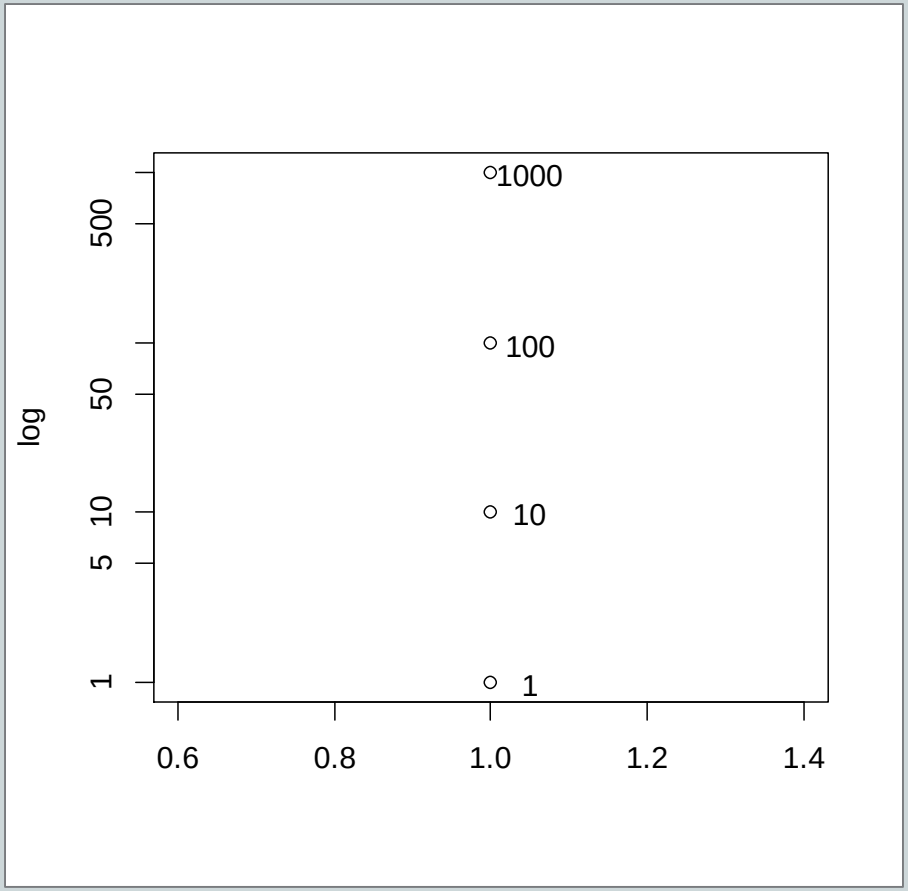
  - Dot product of two vectors divided by the product of their magnitudes
    - Dot product: multiply vectors cell-wise and sum
    - Magnitude: $\|X\| = \sqrt{x_1^2 + x_2^2 .. + x_n^2}$

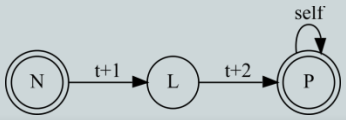


# Log scaling

- Difference between 1 and 10 same as difference between 10 and 100, 100 and 1000, …

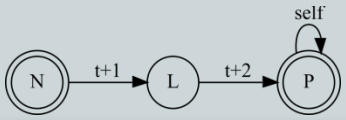
```
from math import log10
print log10(5)
```



# Collinearity

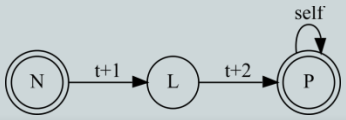
- Consider some terms that go together
- If a document contains these, it's about ..?
  - ***foreign policy***
  - ***lipstick***
  - ***Xi Jinping***
  - ***Revlon***

# Collinearity

- It might be a good idea to combine Revlon and lipstick, or Xi Jinping and …
  - Make some sort of ‘combined word’
  - Or a ‘weighted combined word’:
    - WordX = 75% * f(Xi_Jinping) + 125% * f(policy)

- Dimensionality reduction
- How many words do we want to keep track of?
- Can’t study this in depth for time reasons ☹



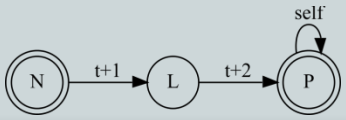
# Handling specificity - TF-IDF

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

- Weight for term *i* in document *j* (or 0 if unattested):

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

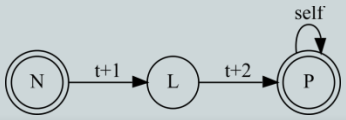
- The IDF weighting for a unique term is maximal: log(N)
- For a term appearing in all documents: log(1) = 0



# So if we know all this

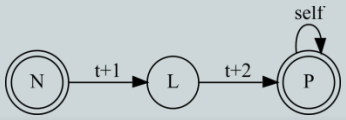
- We can use combined collection and document frequencies!
  - Island w1*3
  - Coast Guard w2*2
  - Navy w3*1
  - U.S. Navy w4*1
  - palm frond w5*1
  - castaway w6*1
  - Honolulu w7*1
  - Fanadik w8*1
  - Strand w9*1

Is this as much about Honolulu as it is about Fanadik?



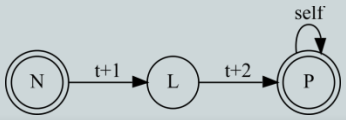
# Classifying documents

- 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 seen is an example of the independent topic-driven probabilities to realize certain words
  - Each document is a mixture of the topics that generated it

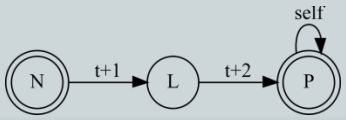


# LDA – a caricature

- 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

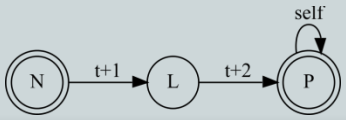
P(Mary|religion) > P(Mary|cooking) > …

Legend:
Religion
Cooking
Engineering
…

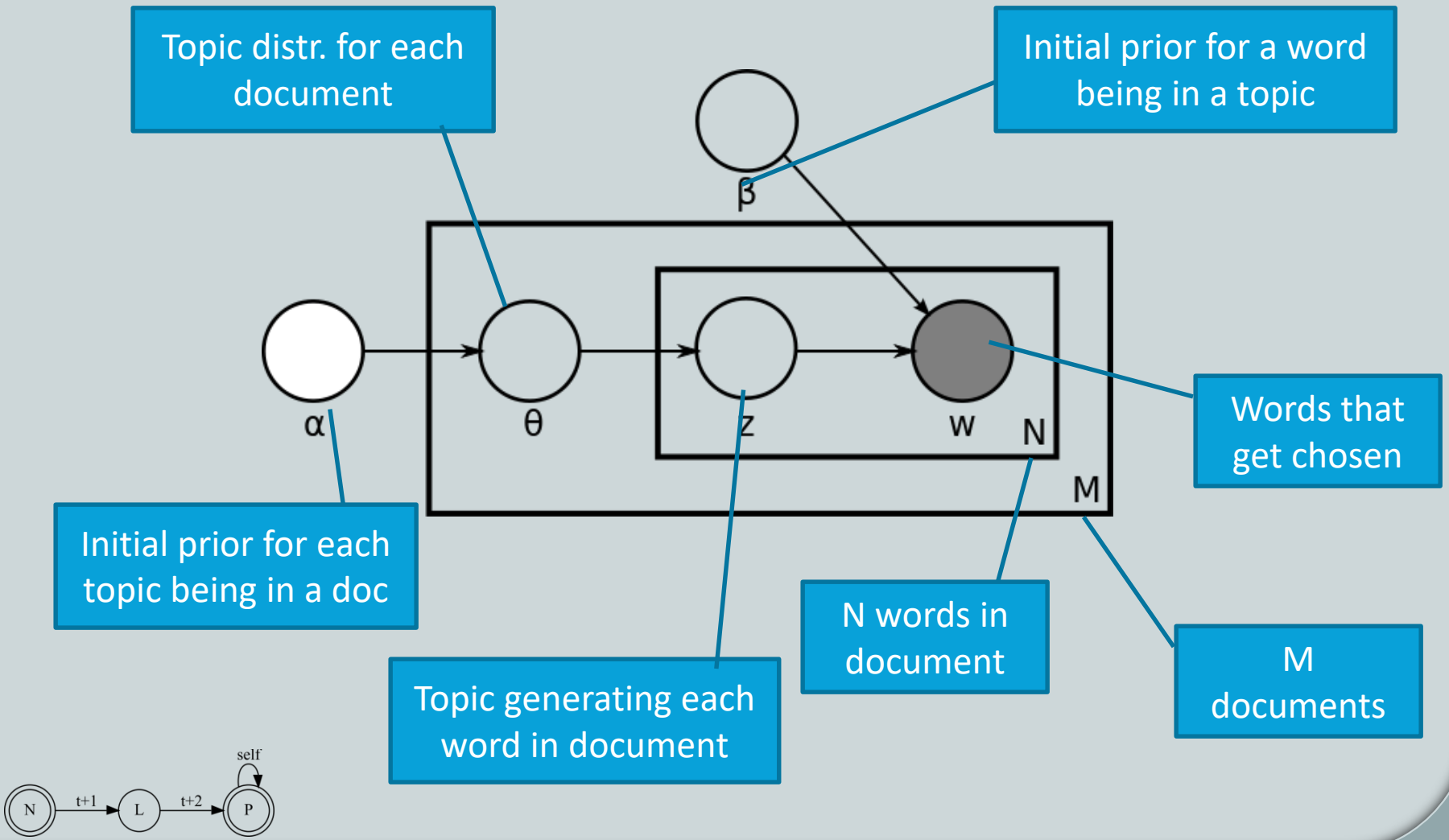


# The 'generative story'

- 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 word is picked at random, based on its probability in that topic – very likely to be 'church', unlikely to be 'pizza' (but possible)

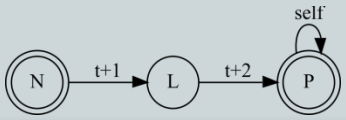


# LDA – more formally



# Inferring latent variables

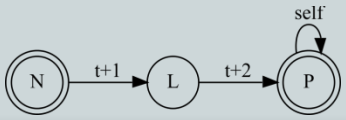
- Now the problem:
  - given the words, some idea of how many topics we might have and what prior distributions are like (incl. likelihood of each word)…
  - Infer the latent variables that generated each document
- Specifically – we want $\vartheta i$ for each document $i$
  - Because if we know what 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 $\vartheta i$
  - 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 $\vartheta$, $\alpha$ and $\beta$
  - Run through data again – is this word's topic 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

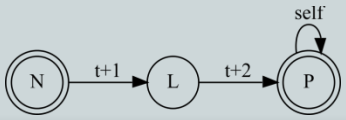
- Further reading: Blei et al. (2003), Grus (2015)



# Library lda

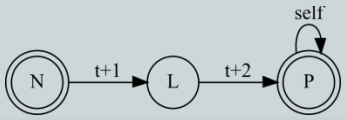
- First we install the lda library from the command line:

  > pip install lda



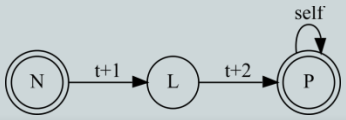
# Imports

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



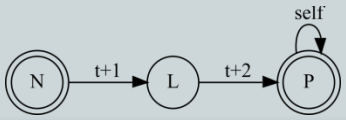
# Example data

```
# 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()
```



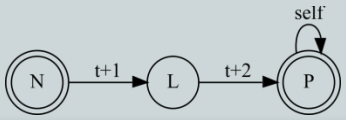
# Testing the data

```
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
```



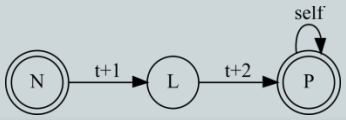
# Fitting the model

```
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

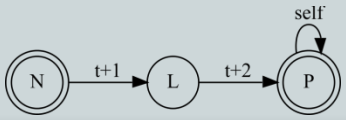
harriman, clinton, u.s, ambassador, paris

pope, vatican, surgery, hospital, rome

died, king, service, funeral, michael

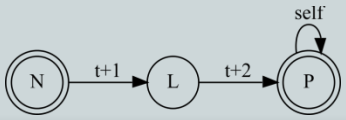
russian, russia, soviet, moscow, communist

…



# 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

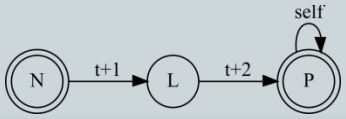
doc1, titled: 1 GERMANY: Historic Dresden church rising from WW2 ashes. DRESDEN, Germany 1996-08-21
Best topic: 4

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

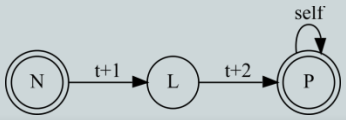
doc4, titled: 4 INDIA: Mother Teresa, slightly stronger, blesses nuns. CALCUTTA 1996-08-25
Best topic: 15

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

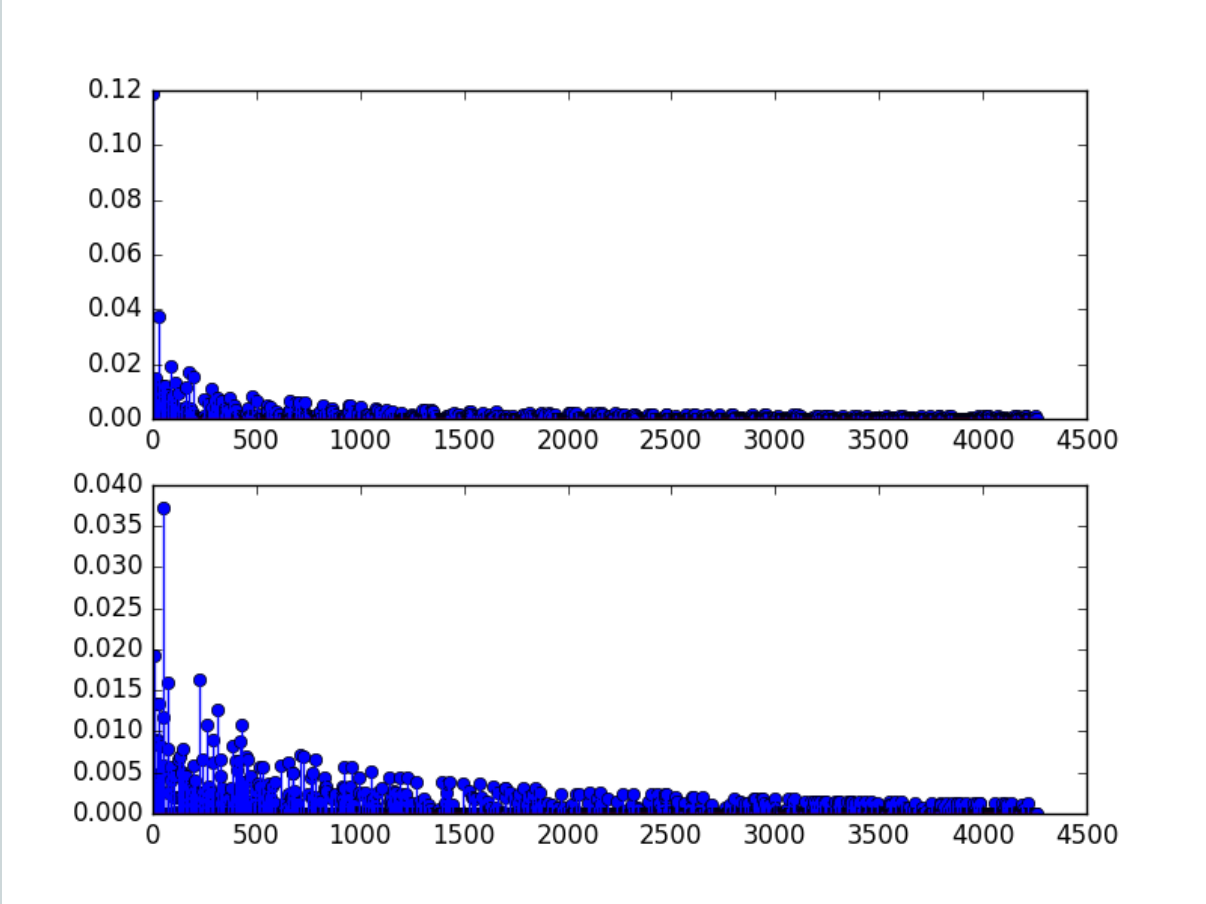


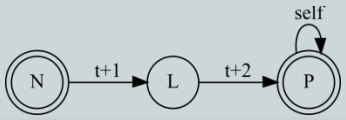
# Plotting word distributions

```
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!

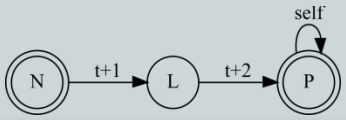


# What else can we read?

- 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
  - Simplified example in Canvas:
    Code > topic_modeling > doc_classification.py



# Preparing for the final

- The final is scheduled for:
  - Thu, 12/16, 12:30-2:30, ICC 116 **(but always check!)**
- The **mock exam** is online
  - We will discuss the questions in class next time
  - Feel free to prepare questions
- The real exam will have the same structure – no need to produce formulas, but important to understand underlying concepts!