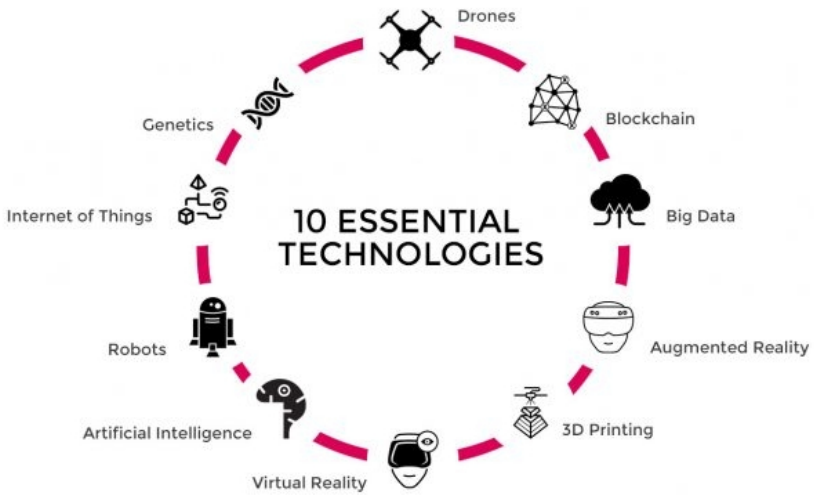
‘how python makes a humanist’s life easier’

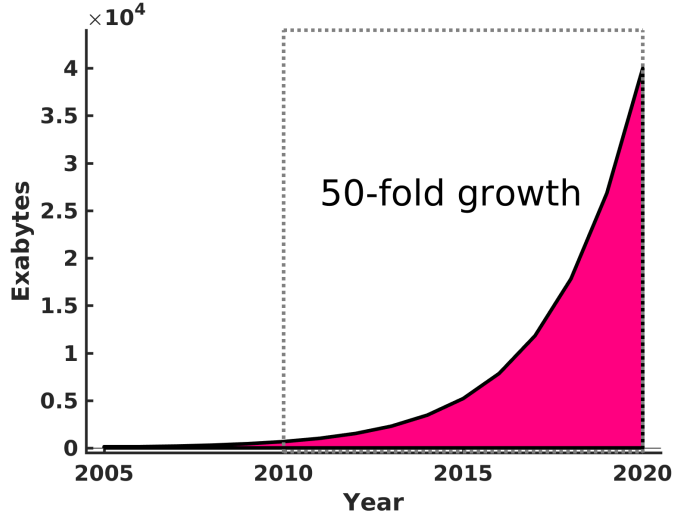
humanities programming
autumn 2016

kristoffer l nielbo
kln@cas.au.dk

DTL|IMC|AU-CAS

IMC
INTERACTING MINDS CENTRE





'Digitization and digital media have generated a rapid proliferation of data that is unprecedented in the history of man. This digital surge is transforming knowledge discovery and understanding in every domain of human inquiry. Digital research, computing, data management, and data-intensive methods will therefore become integral parts of internationally-leading research in the humanities and arts.'

(Digital Arts Strategy 2016)

```
#!/bin/bash

for f in *.pdf
do
  echo "converting: - $f"
  pdftotext $f $f.txt
done

#!/usr/bin/env python

def import_vanilla(datapath):
    docs = []
    files = os.listdir(datapath)
    os.chdir(datapath)
    for file in files:
                ...
    return docs

#!/usr/bin/env julia

srand(1234)
mdl = LDA(docs, 20)
train!(mdl, iter=150, tol=0.0)
    ...
    mat = pairwise(kl_divergence(), theta)

f = figure(1)
h = plot(smooth(mat[10,:],20))
    ...
    savefig(f,figpath,"fig_1.eps")

git commit -am 'ms figure 1 update'

\documentclass[]{interact}
\usepackageepstopdf
...
\includegraphics[scale=.25]{fig_1.eps}
```

topic modeling

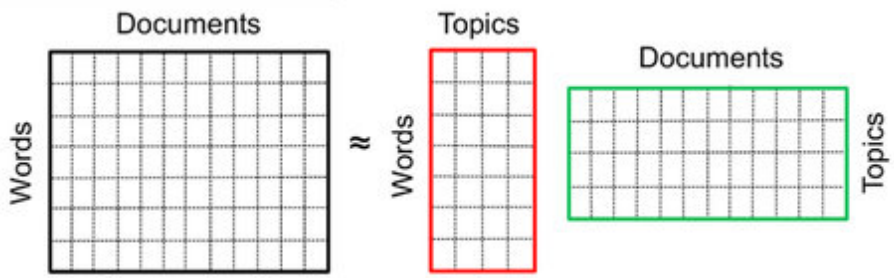
a document is a structured* or non-random collection of words, but who or what is the structuring agent?

to avoid manual modeling, we want method that we can throw ++documents at and then it will sort things out

preferably, the output should exhibit some degree of similarity with human text comprehension

we can extract latent topics that generated the documents by reverse engineering the process *words & doc* $\Rightarrow$ *topics*

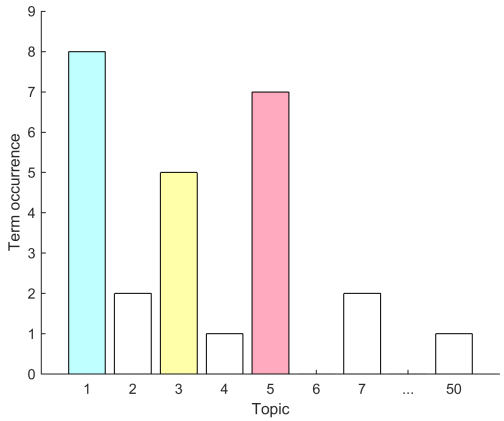
decompose an n-by-d word document matrix into two matrices: a n-by-k word topic matrix and an k-by-d topic document matrix

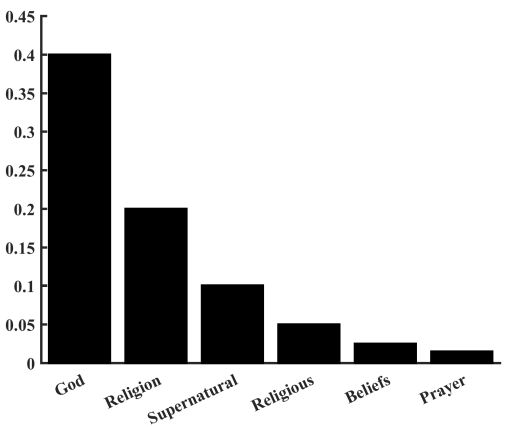


to "generate" a document, choose a distribution over topics, sample a topic (k-by-d matrix), then a word from this topic (n-by-k matrix) and repeat

topic modeling a set of unsupervised mixed models where each document is more or less likely within each clusters

1. **discover** thematic structure
2. **annotate** documents
3. **use** the annotations to visualize, organize, summarize, ...





ABSTRACT—*We present two studies aimed at resolving experimentally whether religion increases prosocial behavior in the anonymous dictator game. Subjects allocated more money to anonymous strangers when God concepts were implicitly activated than when neutral or no concepts were activated. This effect was at least as large as that obtained when concepts associated with secular moral institutions were primed. A trait measure of self-reported religiosity did not seem to be associated with prosocial behavior. We discuss different possible mechanisms that may underlie this effect, focusing on the hypotheses that the religious prime had an ideomotor effect on generosity or that it activated a felt presence of supernatural watchers. We then discuss implications for theories positing religion as a facilitator of the emergence of early large-scale societies of cooperators.*

Many theorists have suggested that the cognitive availability of omniscient and omnipresent supernatural agents has had a dramatic impact on the development of large-scale human societies. The imagined presence of such agents, along with emotional ritual and costly commitment to the social group they govern, may have been the major development that allowed genetically unrelated individuals to interact in cooperative ways (e.g., Atran & Norenzayan, 2004; Irons, 1991; Sosis & Ruffle, 2004). The research reported in this article experimentally investigated this link between two broad classes of culturally widespread phenomena of interest to social science—religious beliefs and cooperative behavior among unrelated strangers.

Although anecdotes documenting religion's prosocial and antisocial effects abound, the empirical literature has produced mixed results regarding religion's role in prosocial behavior.

[redacted]

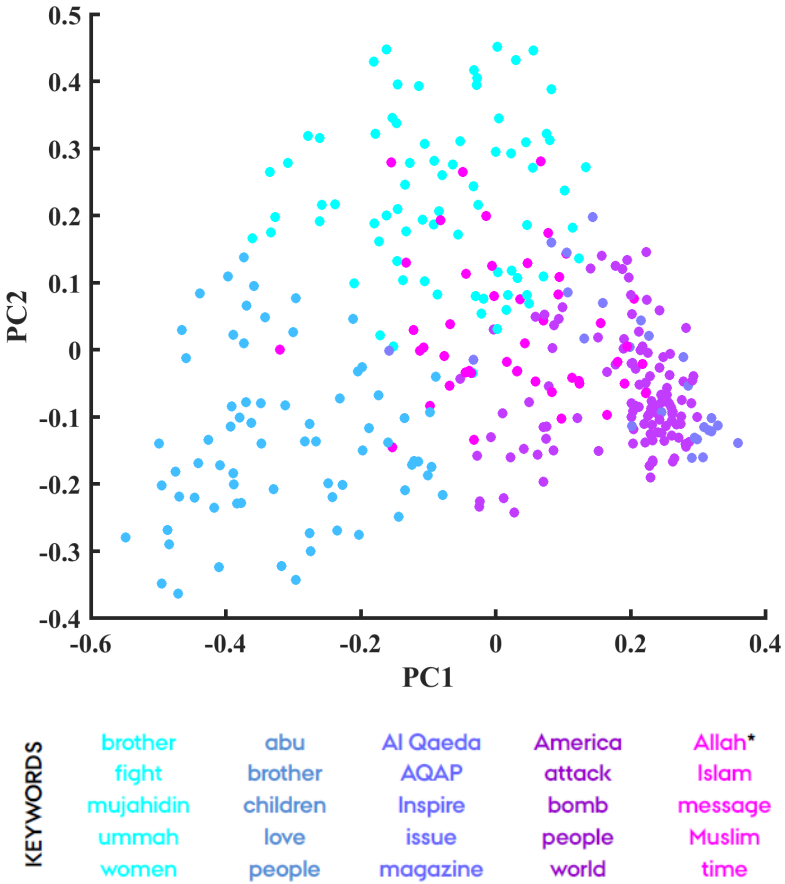
Sosis and Ruffle (2004) examined levels of generosity in an experimental cooperative pool game in religious and secular kibbutzim in Israel and found higher levels of cooperation in the religious ones, and the highest levels among religious men who engaged in daily communal prayer. Batson and his colleagues (Batson et al., 1989; Batson, Schoenrade, & Ventis, 1993) have shown that although religious people report more explicit willingness to care for others than do nonreligious people, controlled laboratory measures of altruistic behavior often fail to corroborate this difference. Furthermore, when studies demonstrate that helpfulness is higher among more devoted people, this finding is typically better explained by egoistic motives such as seeking praise or avoiding guilt, rather than by higher levels of compassion or by a stronger motivation to benefit other people.

However insightful these findings are, research on religion and prosocial behavior has been limited by its overwhelming reliance on correlational designs. If religiosity and prosocial behavior are found to be correlated, it is just as likely that having a prosocial disposition causes one to be religious, or that some third variable such as guilt proneness or dispositional empathy causes both cooperative behavior and religiosity, as that religious beliefs somehow cause prosocial behavior. Only rarely have studies induced supernatural beliefs to examine them as a causal factor. Bering (2003, 2006) inhibited 3-year-old children's tendencies to cheat (i.e., open a "forbidden box") by telling them that an invisible agent ("Princess Alice") was in the room with them. In a different study, college students who were casually told that the ghost of a dead graduate student had been spotted in their private testing room were less willing to cheat on a computerized spatial-reasoning task than were those told nothing (Bering, McLeod, & Shackelford, 2005). These studies suggest that explicit thoughts of supernatural agents curb cheating behavior.

In the research reported here, we examined the effect of God concepts specifically on selfish and prosocial behavior. Our research design was novel in two ways. First, we introduced an

IMC
INTERACTING MINDS CENTRE

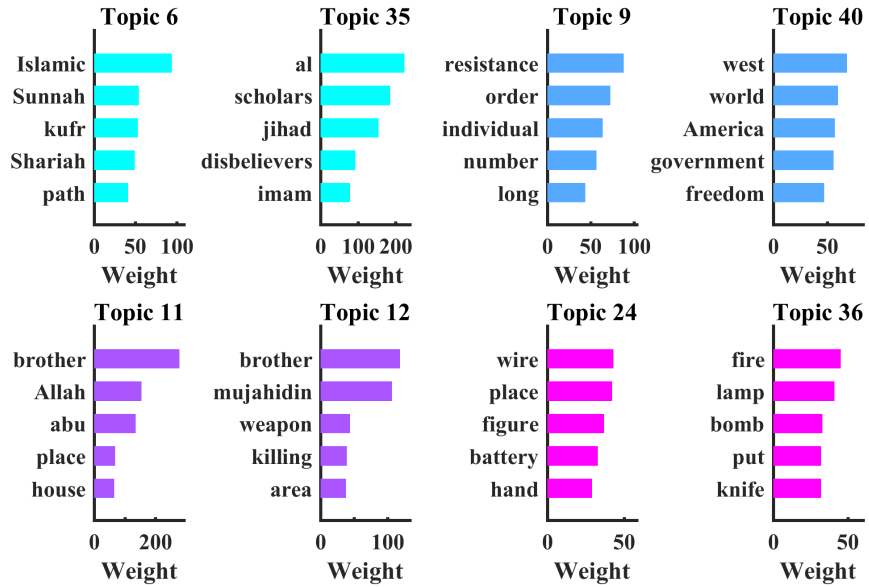
jihadist macroscope

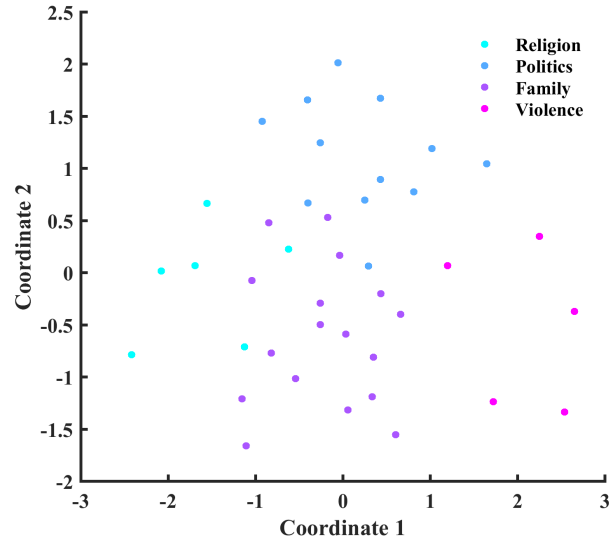


trying to get a handle on islamist propaganda targeting western youth

classical document (hard) clustering and low dimensional projections

model neither a good representation of discourse nor it performing well





lda + to the rescue, $Z_{d,n}$ is the topic for the n^{th} word in document d

model the multitude of topics in documents and compare their similarity

estimation of (hyper-) parameters is computationally demanding, so speed is important

```
1  def vanilla_folder(datapath):
2      docs = []
3      files = sorted(os.listdir(datapath), key = numericalsort)
4      os.chdir(datapath)
5      for file in files:
6          print "file import: " + file
7          with io.open(file,"r",encoding = "utf8") as f:
8              text = f.read()
9              text = unicodedata.normalize("NFKD", text)
10             text = re.sub(r"\d+", " ",text)
11             docs.append(re.sub(r"\W+", " ",text))
12     return docs
13
14 def vanilla_tokenize(docs):
15     unigrams = [[w for w in doc.lower().split()] for doc in docs]
16     return unigrams
```

```
from gensim.utils import chunkize
def vanilla_chunk(unigrams,n):
    chunks = []
    for doc in unigrams:
        clen = len(doc)/n
        for chunk in chunkize(doc,clen):
            chunks.append(chunk)
    return chunks

from collections import defaultdict
import numpy as np
def vanilla_prune(unigrams,mxper,mnper):
    frequency = defaultdict(int)
    for doc in unigrams:
        for unigram in doc:
            frequency[unigram] += 1
    freqs = [val for val in frequency.values()]
    mx = np.percentile(freqs, mxper)
    mn = np.percentile(freqs, mnper)
    unigrams_prune = [[unigram for unigram in doc if (frequency[unigram] > mn and \
            frequency[unigram] <= mx)] for doc in unigrams]
    return unigrams_prune
```

```
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag

# change from treebank to wordnet POS tags
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith("J"):
        return wordnet.ADJ
    elif treebank_tag.startswith("V"):
        return wordnet.VERB
    elif treebank_tag.startswith("N"):
        return wordnet.NOUN
    elif treebank_tag.startswith("R"):
        return wordnet.ADV
    else:
        return wordnet.NOUN

def vanilla_lemmatizer(unigrams):
    wordnet_lemmatizer = WordNetLemmatizer()
    unigrams_lemma = unigrams
    for i, _ in enumerate(unigrams):
        tmp = pos_tag(unigrams[i])
        for ii, _ in enumerate(tmp):
            unigrams_lemma[i][ii] = wordnet_lemmatizer.lemmatize(tmp[ii][0],\
                    get_wordnet_pos(tmp[ii][1]))
    return unigrams_lemma
```

```
texts = vanilla_folder(datapath)
tokens = vanilla_tokenize(texts)
chunks100 = vanilla_chunk(tokens,100)
prune98 = vanilla_prune(chunks100,98,0)
lemmanoun = vanilla_lemmatizer(prune98)

import numpy as np
from gensim import corpora, models

dictionary = corpora.Dictionary(lemmanoun)
corpus = [dictionary.doc2bow(chunk) for chunk in lemmanoun]
fixed_seed = 1234
np.random.seed(fixed_seed)
k = 20
mdl = models.LdaModel(corpus, id2word=dictionary, num_topics=k)

print 'Z:'
for i in range(k): print "topic "+`i+1`+":" + "\n"+`mdl.show_topic(i)`+"\n"

print "Theta:"
for i in range(k): print `lemmanoun[i][0:19]`+"\n"+`mdl.get_document_topics(corpus[i],0)`+"\n"
```

latent dirichlet allocation

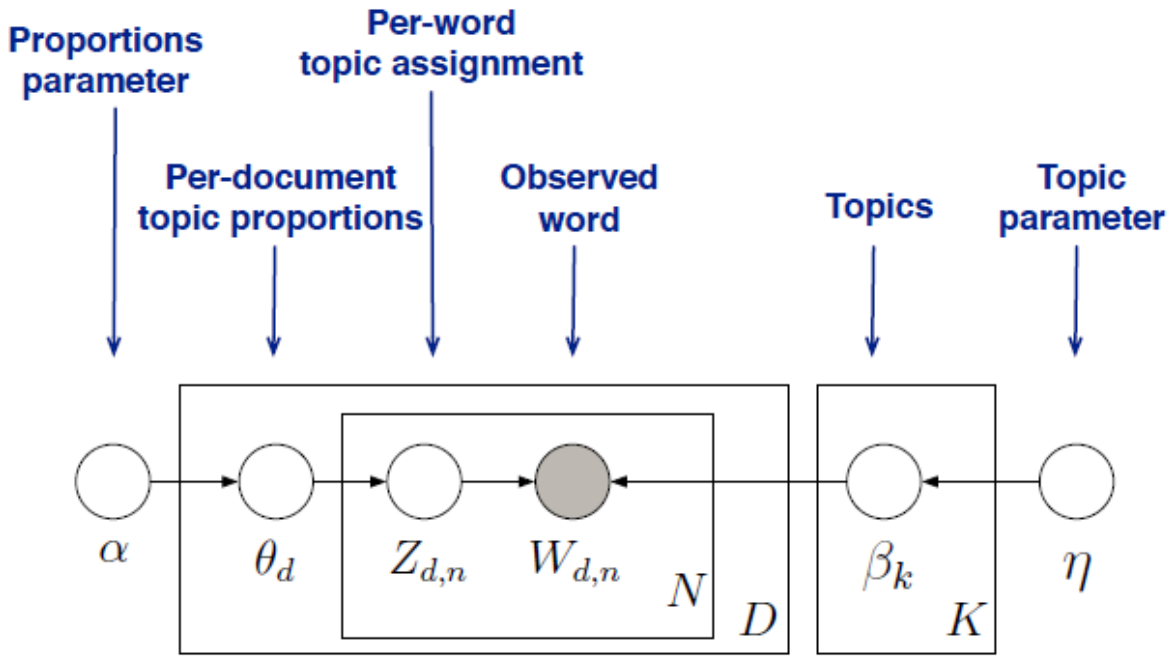
simple three-level bayesian model that uses expectation-maximization or Gibbs sampling to fit the model

goals of LDA $\sim$ competing sparsity

1. for each *document*, allocate its words to as *few topics* as possible
2. for each *topic*, assign high probability to as *few words* as possible

all words in a document have a probability under each topic (1), but to cover all a document's words, we must assign many topics to is (2)

$\Rightarrow$ trade off between goals results in tightly co-occurring words within each topic

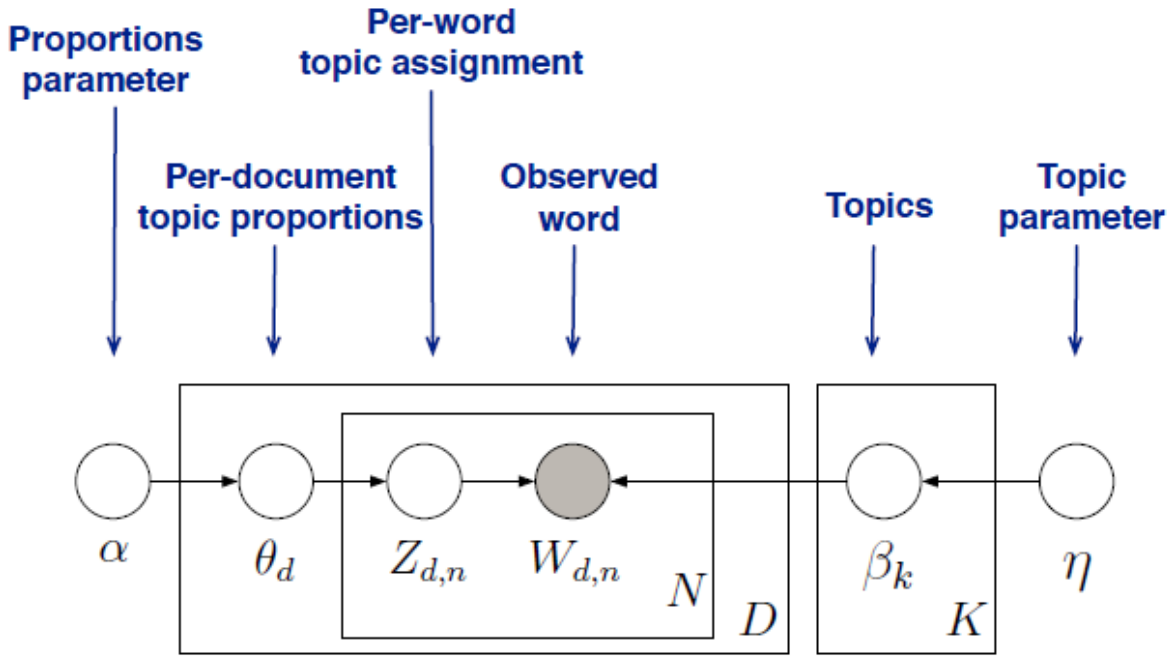


the joint distribution defines a posterior probability: $p(\theta, z, \beta \mid w)$

from a collection of documents we can infer

- per-word topic assignment $Z_{d,n}$
- per-document topic proportions θ_d
- per-corpus topic distributions β_k

use posterior expectations to perform given task

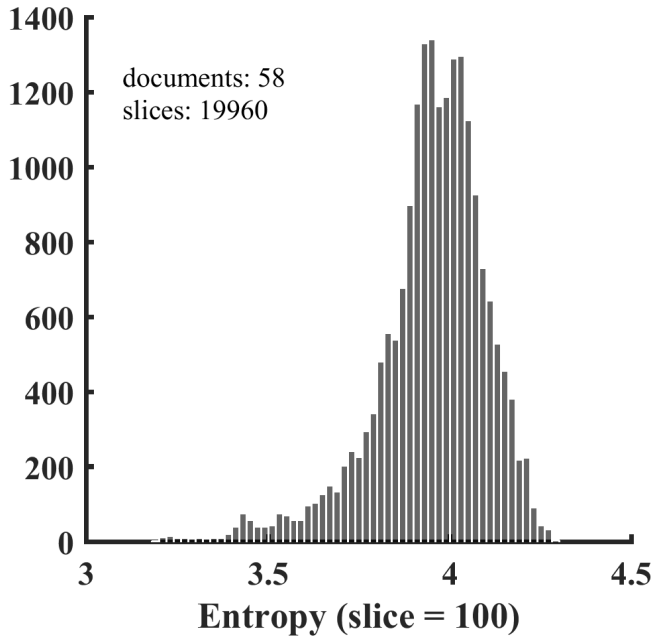


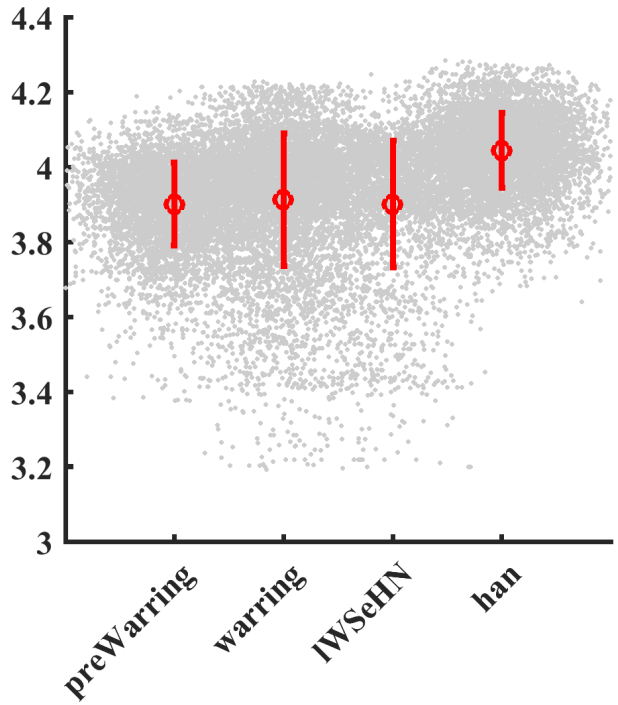
k determines the model's granularity and can be estimated using model perplexity

low α value places more weight on documents composed of only a few dominant topics (high value will return more relatively dominant topics) $\rightarrow \frac{50}{k}$

a low η/β value places more weight on having each topic composed of only a few dominant words $\rightarrow$ 0.01

chinese evolution



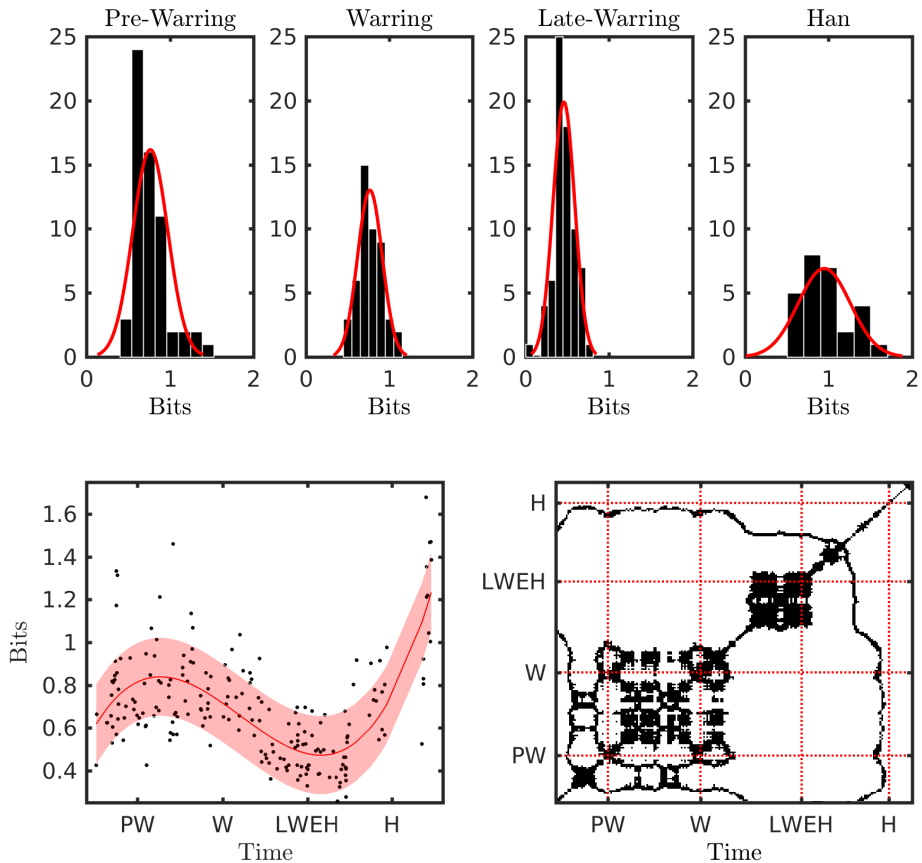


"a rose is a rose is a rose" is less lexically dense than "a rose is red and thorny"

lexical density $\sim$ text predictability: $H(X) = \sum_{i=1}^{n} p_i \log_2 p_i$

$$H(\text{a rose is a rose is a rose}) < H(\text{a rose is red and thorny})$$

$$H(\text{a rose is a rose is a rose}) = H(\text{erea oisasessar oiors})$$



lda + 🐍 to the rescue θ_i probability distribution of k latent variables in document i

disruption between document is the relative entropy: $D_{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$

```
import os
import numpy as np
import matplotlib.pyplot as plt

os.chdir("/home/kln")
docmat = np.genfromtxt("data/theta.csv", delimiter=",")

def kl(p, q):
    p = np.asarray(p, dtype=np.float)
    q = np.asarray(q, dtype=np.float)
    return np.sum(np.where(p != 0,(p-q) * np.log10(p / q), 0))

def movavg(vect, n = 5) :
    ret = np.cumsum(vect, dtype = float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n

# kld for doc_n to mean doc_1:doc_n-1
kld = np.zeros(len(docmat))
for i in range(1,len(docmat)):
    submat = docmat[0:i,]
    tmp = np.zeros(len(submat))
    for ii in range(len(submat)):
        tmp[ii] = kl(submat[ii,],docmat[i,])
    kld[i] = np.mean(tmp)

kld_smooth = movavg(kld-np.mean(kld), n=100)

plt.figure(1)
h = plt.plot(kld_smooth)
plt.setp(h, 'color', 'r', 'linewidth', 2.0)
plt.xlabel('Time Index')
plt.ylabel('Bits')
plt.axis([0, len(kld_smooth),min(kld_smooth)*1.1,max((kld_smooth))*1.1])
plt.grid(True)
plt.show()
```

so did 🐍 make my life easier?

case #1 automatized and efficient way for managing the *data surge*

case #2 versatile (general-purpose) programming language that offers ++packages