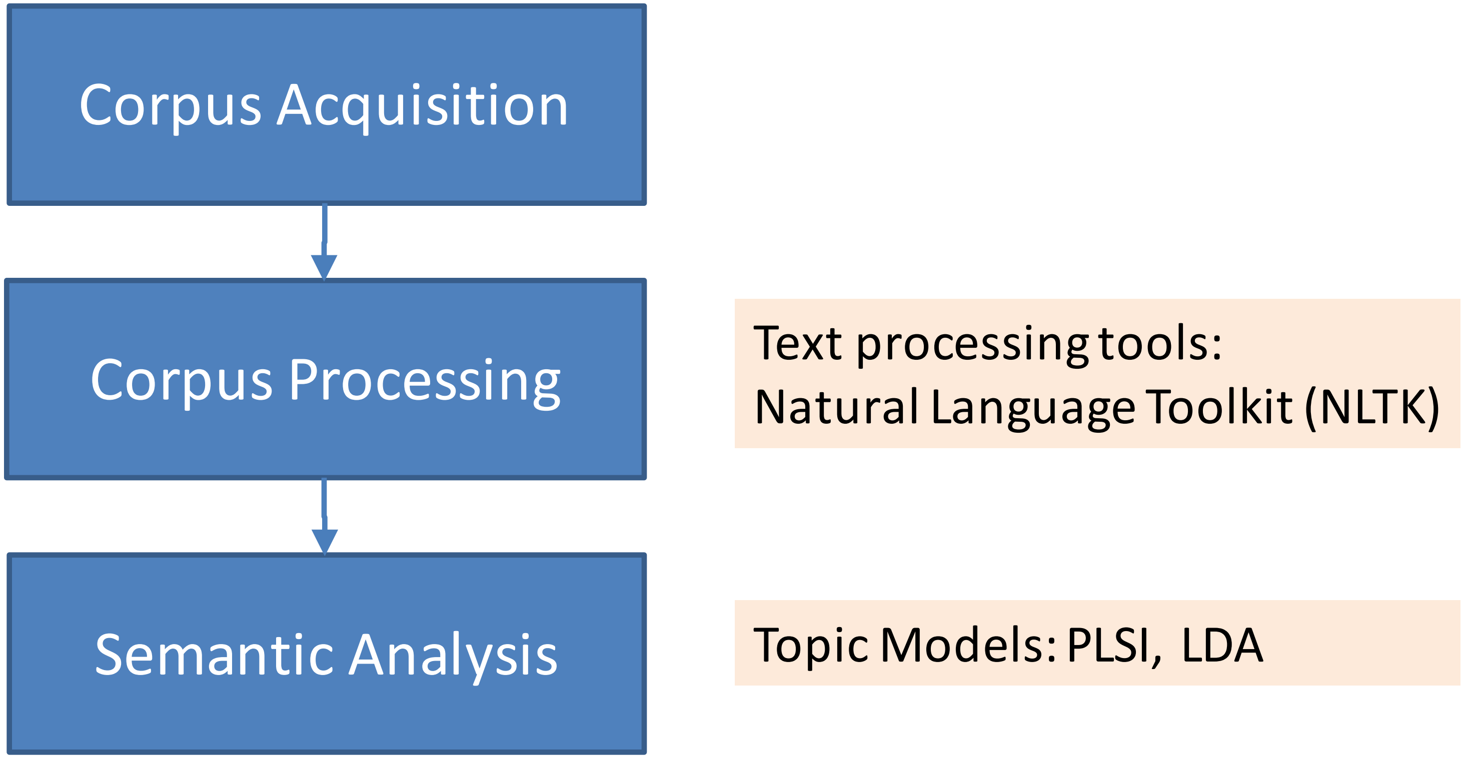
Topic Modeling

Vanessa Gómez Verdejo
Jesús Cid Sueiro

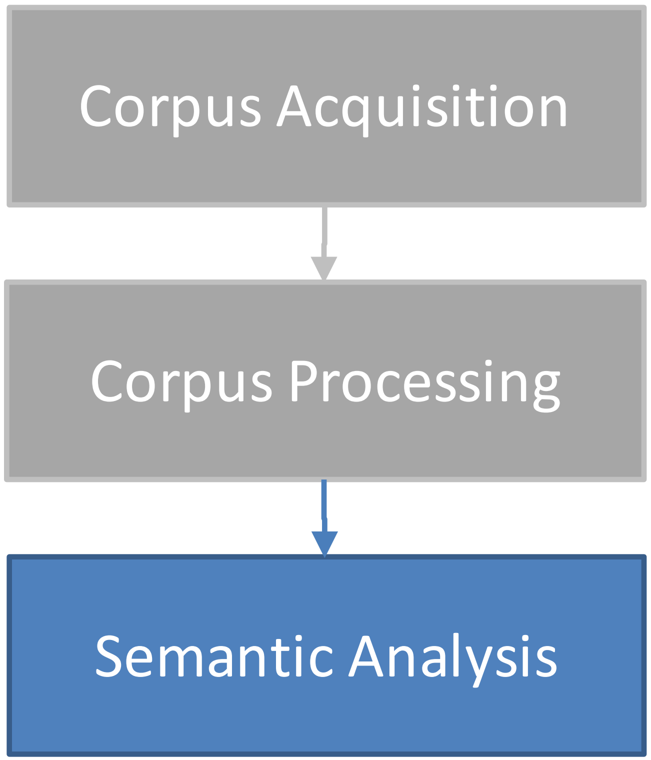
Content

- 1. Topic Models
- 2. Latent Semantic Indexing
- 3. Latent Dirichlet Allocation

1. Topic Models



Semantic Analysis



Text processing tools:
Natural Language Toolkit (NLTK)

Topic Models: PLSI, LDA

Topic models

- Topic Models attempt to uncover the underlying semantic structure of a document corpus by identifying recurring patterns of terms (topics).
- Topic models are models for bags-of-words:
  - do not parse sentences
  - do not care about word order, and
  - do not “understand” grammar or syntax
- Topic models are useful on their own to build visualizations and explore data. They are also very useful as an intermediate step in many other tasks.

Topic modelling tools

- Gensim
  - Developed by Radim Řehůřek.
  - Topics and transformations: Gensim includes
    - BOW
    - TF-IDF
    - Latent Semantic Indexing, LSA/LSI
    - Latent Dirichlet Allocation, LDA

Gensim

- Instalation:

```
pip install gensim
easy_install gensim
```

- Data Structures
  – Corpus: list of documents
  – Document: list of words

Working with Gensim

1. Import tools.

```
from gensim import corpora
```

2. Represent the words by ids (integer) → create a dictionary

```
D = corpora.Dictionary(docs)
```

3. Vectorize the documents: create bow.

```
bow = [D.doc2bow(doc) for doc in docs]
```

   – Gensim has efficient implementations for long corpus (work document to document): http://radimrehurek.com/gensim/tut1.html

Working with Gensim

4. Compute tf-idf values.

```
from gensim import models
# 1-- initialize a model
tfidf = models.TfidfModel(corpus_bow)
```

– From now on, tfidf can be used to convert any vector from the old representation (bow integer counts) to the new one (TfIdf real-valued weights):

```
doc_bow = [(0, 1), (1, 1)]
# 2-- transform  a new vector
tfidf[doc_bow]
```

– Or to apply a transformation to a whole corpus:

```
corpus_tfidf = tfidf[corpus_bow]
```

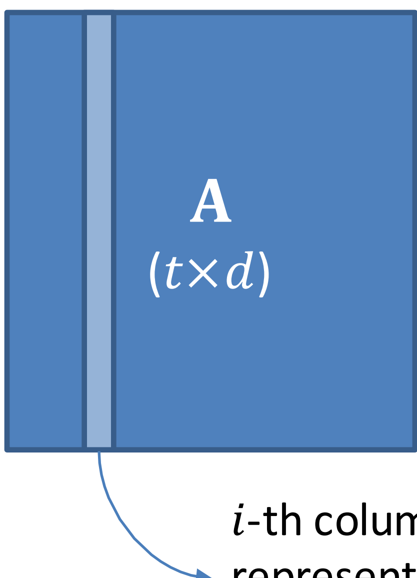
2. Latent Semantic Indexing

- It transforms documents from either
  - bag-of-words, or
  - (preferably) TfIdf-weighted space

  into a latent space of a lower dimensionality.
- LSI is able to correlate semantically related terms that are latent in a collection of text
- LSI uses example documents to establish the conceptual basis for each category.
- LSI overcomes two of the most problematic constraints of Boolean keyword queries: multiple words that have similar meanings (synonymy) and words that have more than one meaning (polysemy).

Latent Semantic Analysis/Indexing

- The starting point of LSI is the TF-IDF matrix, $\mathbf{A}$, with size $(t \times d)$,
  - t is the number of terms (tokens), and
  - d is the number of documents



i-th column: TF-IDF representation of document i

Latent Semantic Analysis/Indexing

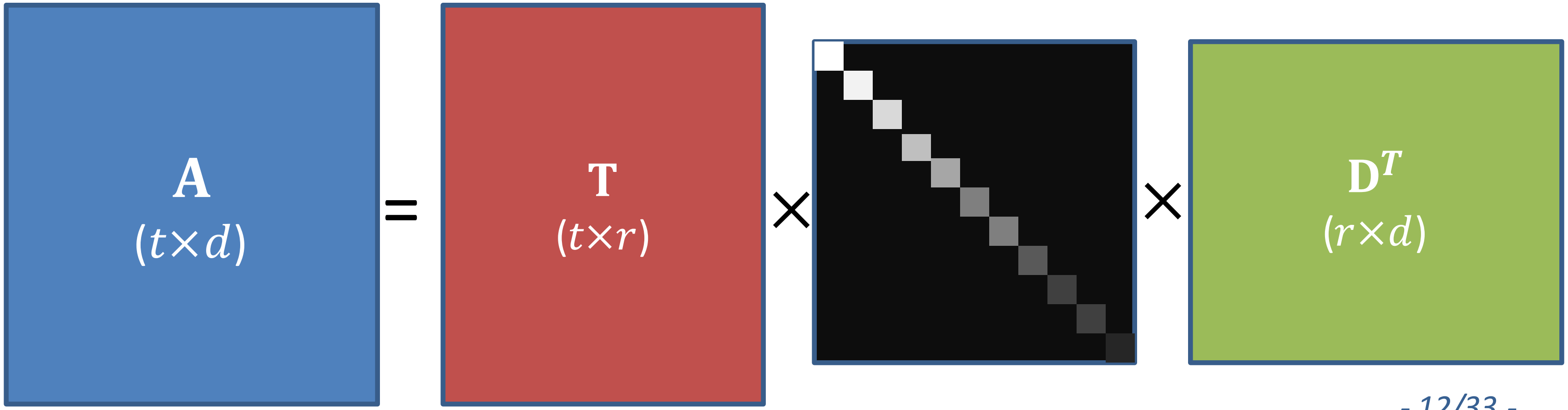
- LSI computes the term and document vector spaces by means of the singular value decomposition (SVD) of matrix, $\mathbf{A}$ $(t \times d)$, with rank r, into the product of 3 matrices:

$$\mathbf{A} = \mathbf{TSD}^T$$

  - $\mathbf{T}$ $(t \times r)$: Unitary concept vector matrix: $\mathbf{T}^T\mathbf{T} = \mathbf{I}_r$
  - $\mathbf{S}$ $(r \times r)$: Diagonal matrix of singular values

$$s_{11} > s_{22} > \cdots > s_{rr} > 0$$

  - $\mathbf{D}$ $(d \times r)$: Unitary concept-document matrix: $\mathbf{D}^T\mathbf{D} = \mathbf{I}_r$



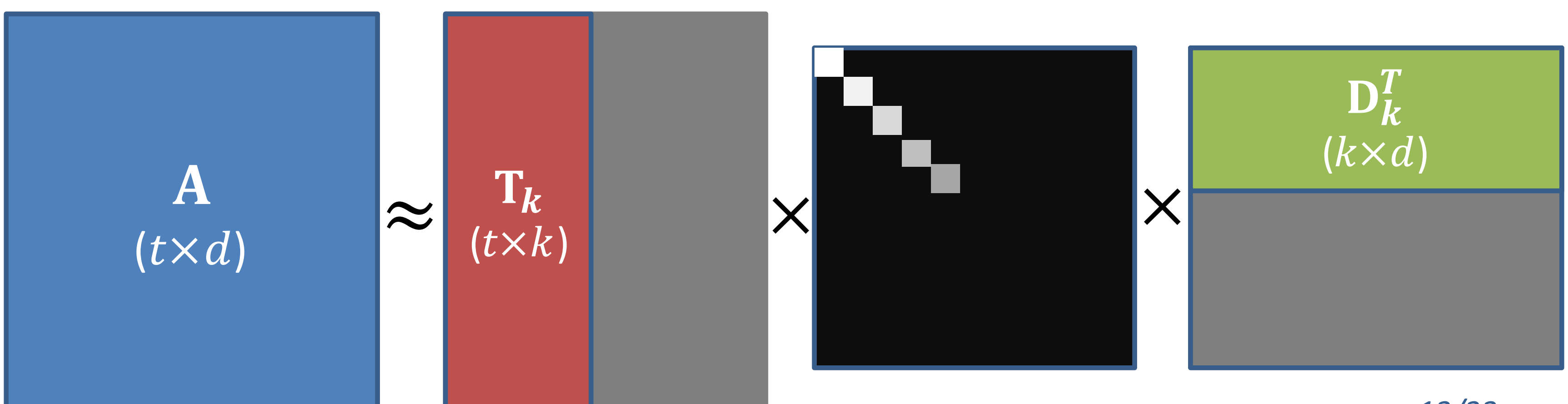
Latent Semantic Analysis/Indexing

- LSI approximates $\mathbf{A}$ using a reduced number (k) of concepts (the "topics" in LSI), by ignoring the smallest singular values

$$s_{k+1,k+1} \approx 0, \dots, s_{r,r} \approx 0 \quad \Rightarrow$$

- Thus, $\mathbf{A}$, is approximated as :

$$\mathbf{A} \approx \mathbf{T}_k \mathbf{S}_k \mathbf{D}_k^T$$



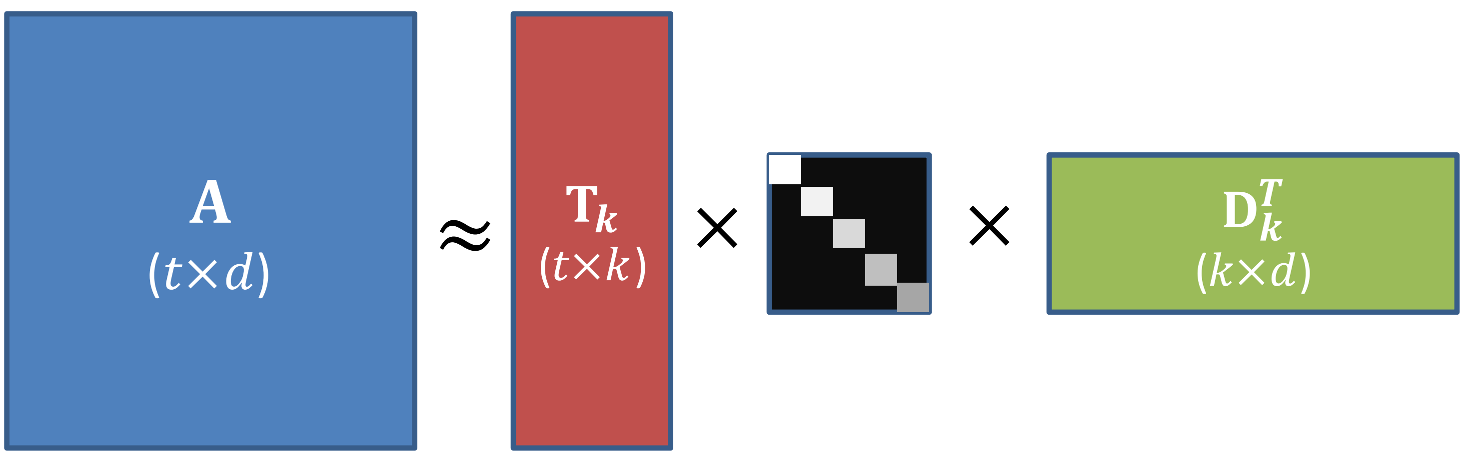
Latent Semantic Analysis/Indexing

- LSI approximates $\mathbf{A}$ using a reduced number (k) of concepts (the “topics” in LSI), by ignoring the smallest singular values

$$s_{k+1,k+1} \approx 0, \ldots, s_{r,r} \approx 0 \quad \Rightarrow$$

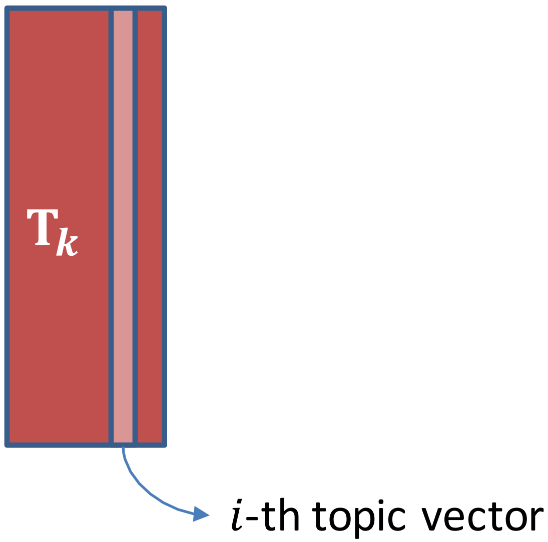
- Thus, $\mathbf{A}$, is approximated as :

$$\mathbf{A} \approx \mathbf{T}_k \mathbf{S}_k \mathbf{D}_k^T$$



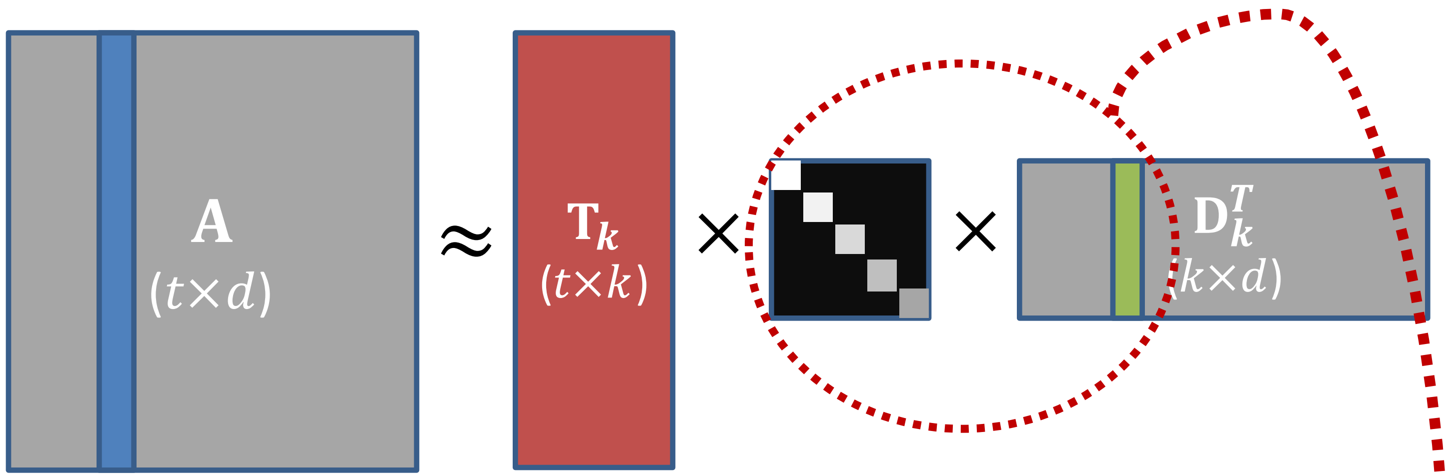
Latent Semantic Analysis/Indexing

- $\mathbf{T}_k$ the topic matrix.
  - Column n in $\mathbf{T}_k$ represents topic n as a vector of weights, one per token.

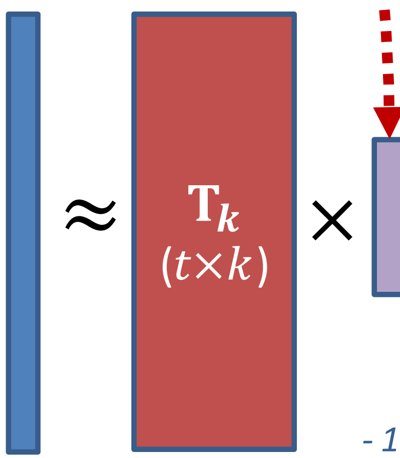


Latent Semantic Analysis/Indexing

- Note that the n-th column of $\mathbf{A}$ depends on the n-th column of $\mathbf{D}_k^T$ only.



- This shows that LSI approximates each document vector in $\mathbf{A}$ is approximated as a linear combination of the topic vectors.



LSA-LSI in Gensim

- Steps to transform our Tf-Idf corpus via LSI into a latent 2-D space

```
# Initialize an LSI transformation
lsi = models.LsiModel(corpus_tfidf,
    id2word=dictionary, num_topics=2)

# On real corpora, target dimensionality of
# 200–500 is recommended as a “golden standard”
# Create a double wrapper over the original
# corpus bow → tfidf → fold-in-lsi
corpus_lsi = lsi[corpus_tfidf]
```

- It allows incremental updates:

```
lsi.add_documents(another_tfidf_corpus)
```

LSA-LSI

- Analyzing the topics:

```
# both bow → tfidf and tfidf → lsi
# transformations are actually executed here,
# on the fly
>> lsi.print_topics(2)
topic #0(1.594): -0.703*"trees" + -0.538*"graph"
+ -0.402*"minors" +…
topic #1(1.476): -0.460*"system" + -0.373*"user"
+ -0.332*"eps" +….
```

- "trees", "graph" and "minors" are all related words (and contribute the most to the direction of the first topic)

LSA-LSI

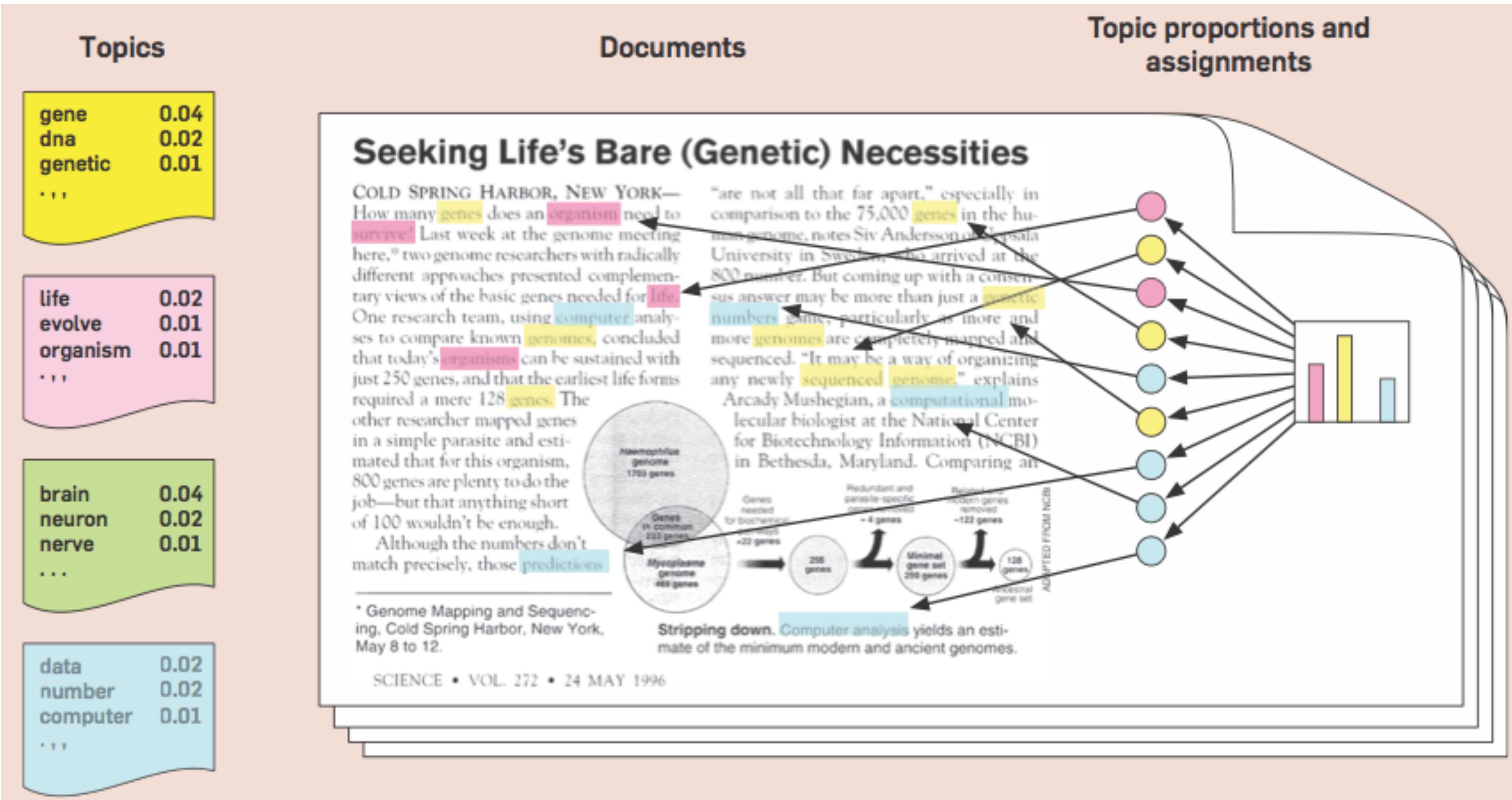
- Analyzing document representation over the topics:

```
>> for doc in corpus_lsi:
       print(doc)

# "The intersection graph of paths in trees"
[(0, -0.877), (1, -0.168)]

# "System and human system engineering testing of
# EPS”
[(0, -0.076), (1, 0.632)]
```

3. Latent Dirichlet Allocation (LDA)

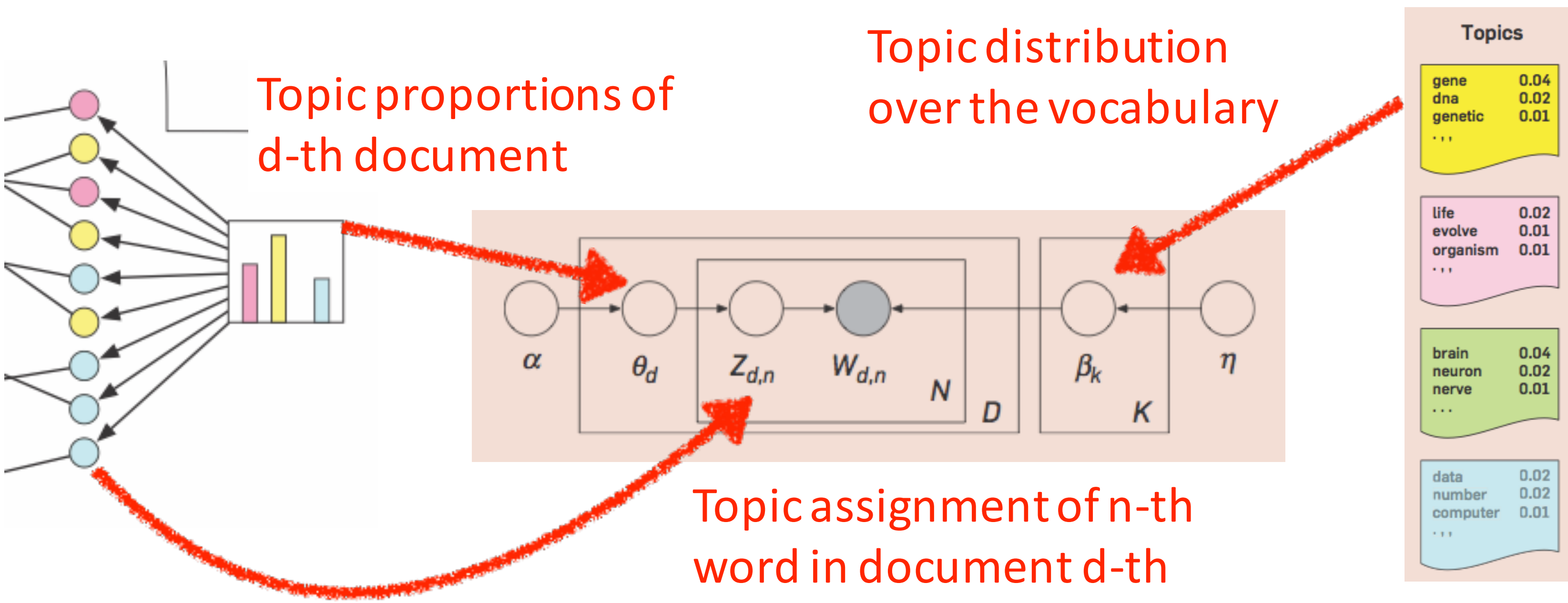
- LDA is another transformation from bag-of-words counts into a topic space of lower dimensionality.
- LDA is a probabilistic extension of LSA, so LDA’s topics can be interpreted as probability distributions over words.
- These distributions are inferred automatically from a training corpus.
- Documents are in turn interpreted as a (soft) mixture of these topics (again, just like LSA).

Latent Dirichlet Allocation

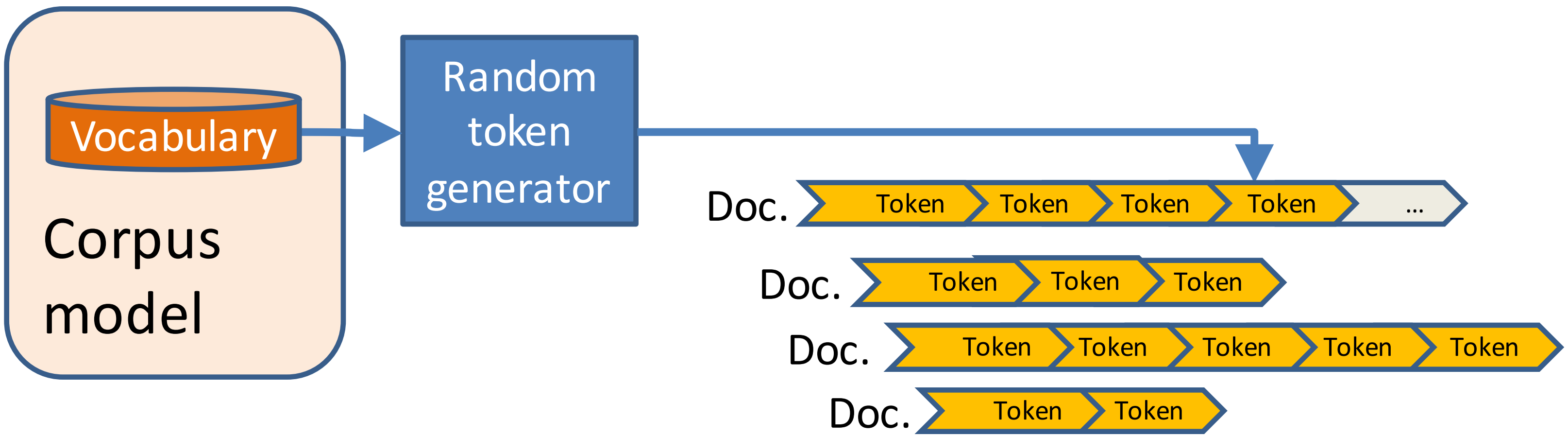


Understanding LDA

- LDA is a generative probabilistic model:
  - it assumes that documents have been generated according to some probability model with some unknown parameters an certain hidden variables
  - The corpus data is used to inferring the topic structure (hidden variables) from the words of documents (observed variables)

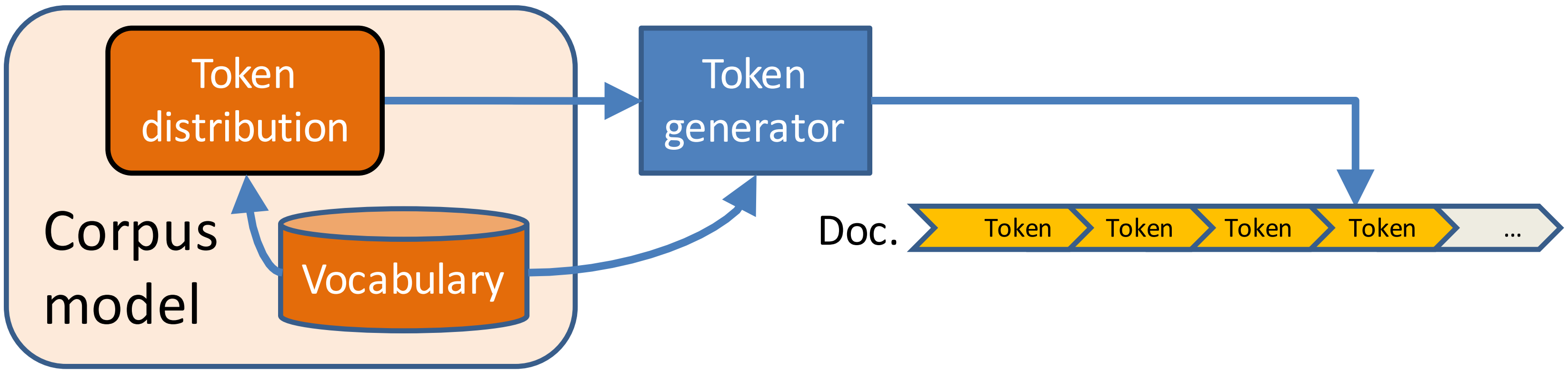


A naïve corpus generator:



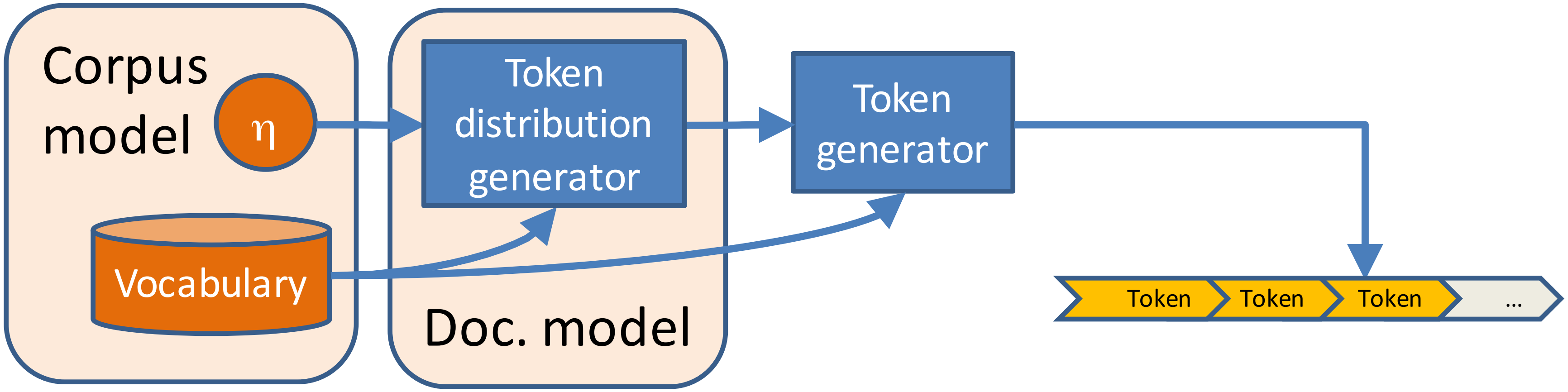
- Inference:
  - Given the corpus, obtain the vocabulary.
- Limitations:
  - In real documents, token proportions are far from uniform.

An homogeneous corpus generator:



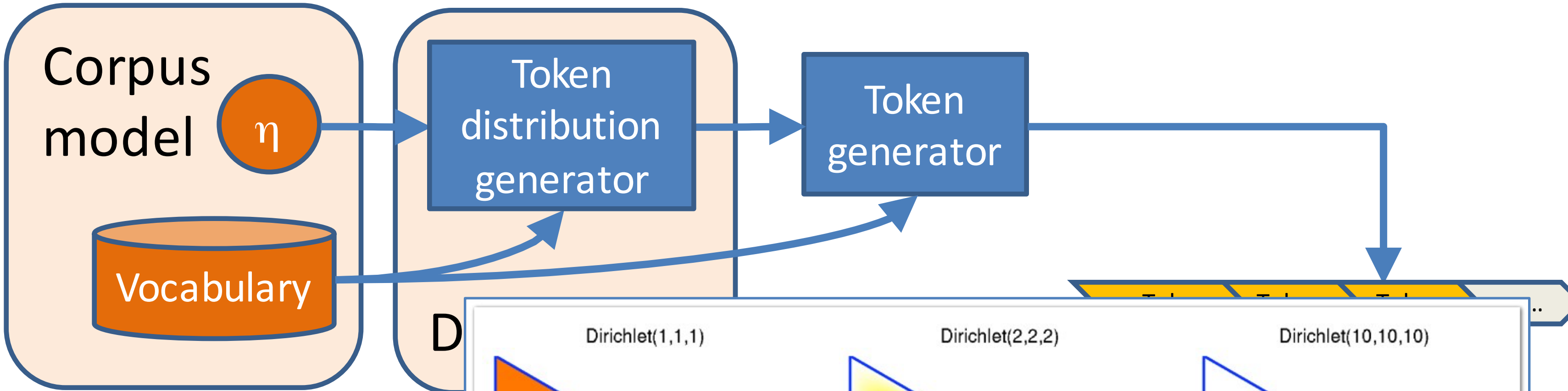
- Advantages:
  - Differentiated token proportions.
- Inference (for a given corpus):
  - Compute the vocabulary and estimate the token distribution.
  - The token distribution can be easily estimated from token frequencies measured over the whole corpus.
- Limitations:
  - Real corpora are heterogeneous: each document use to obey a different token distribution.

An heterogeneous corpus generator:



- The token distribution is a random variable, $\boldsymbol{\beta}$
- Each document is generated with a different token distribution.
- Each token distribution is an outcome of a (symmetric) Dirichlet distribution with *concentration parameters* η.
  - $P(\boldsymbol{\beta}|\eta) = \frac{1}{B(\eta)} \prod_{i=1}^{n} \beta_i{}^{\eta-1}$
  - $B(\eta)$ is a normalizing constant: $B(\boldsymbol{\eta}) = \frac{\Gamma(\eta)^{\boldsymbol{n}}}{\Gamma(n\eta)}$, where Γ is the Gamma function.

An heterogeneous corpus generator:



- Example: Dirichlet distributions for a vocabulary of size 3.
- For large η, probabilities concentrate in a few terms.

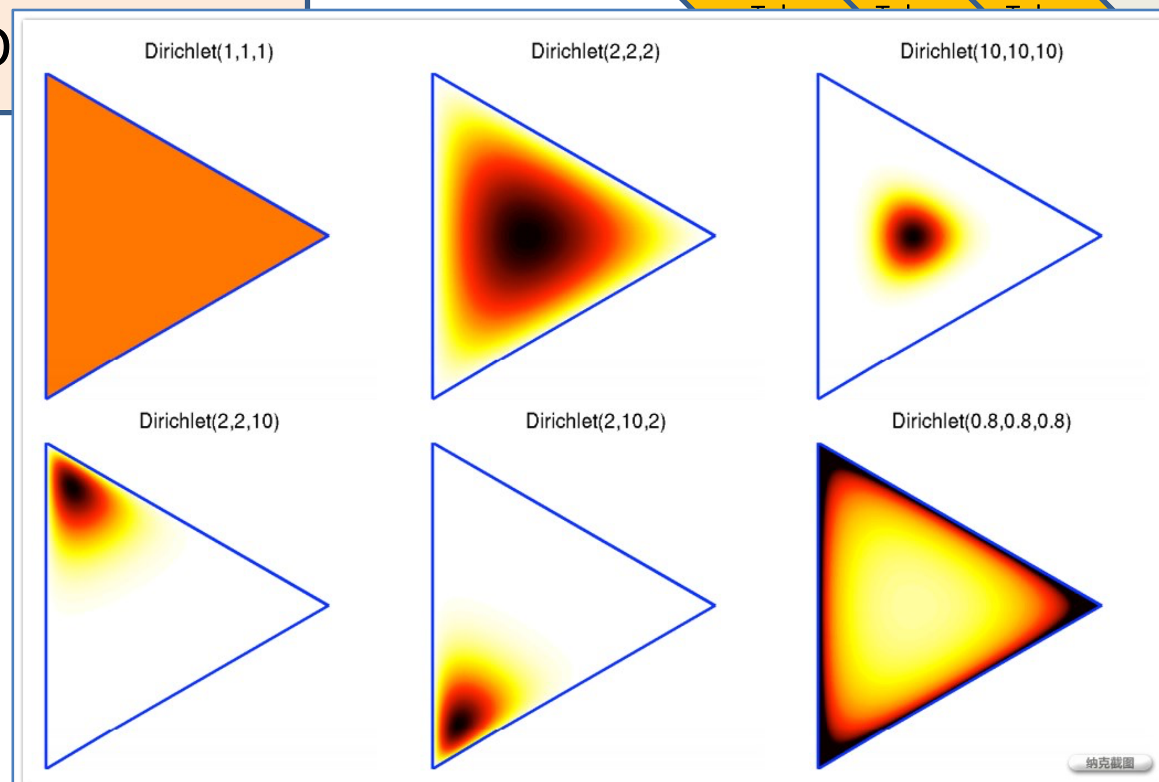
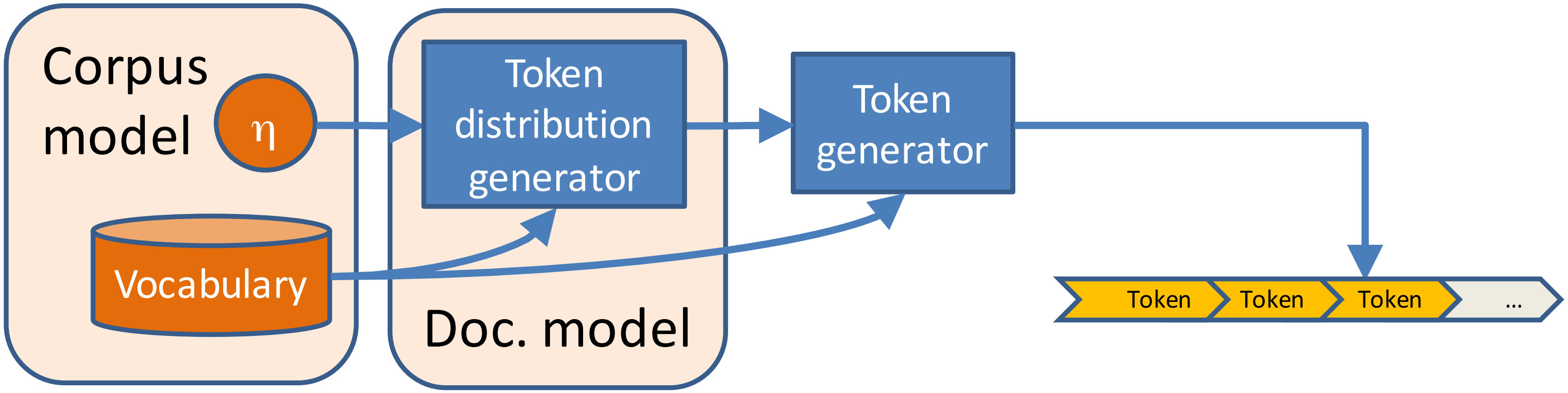


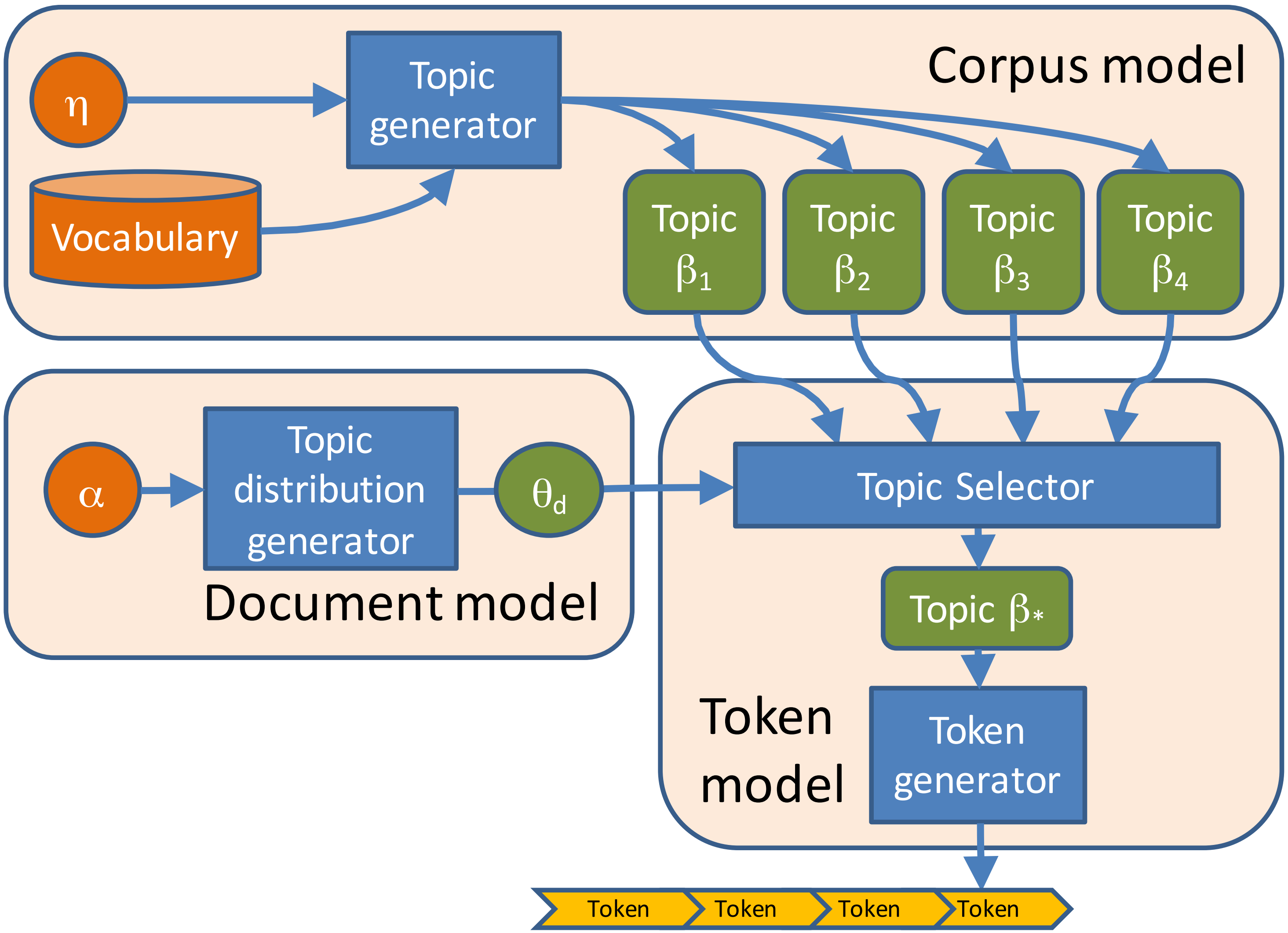
Image taken from http://www.cnblogs.com/huashiyiqike/articles/3232082.html

An heterogeneous corpus generator:



- Inference (given the corpus and η):
  - Collect vocabulary and estimate the token distribution for each topic.
- Advantages:
  - Each document has different token proportions.
  - Concentration parameter can be tuned to generate sparse token distributions, which generate documents that use only a portion of words in the vocabulary. This is realistic in many corpora.
- Limitations:
  - Real corpora have some topic structure. Token distributions for documents from different topics tend to be substantially different.

The LDA corpus generator:



The LDA corpus generator:

- Topic generator:
  - Generates tokens distributions by means of a Dirichlet distribution with concentration parameter η.
  - Large η ➔ topic distributions will be concentrated in a few distinct tokens ➔ documents generated by a single topic will contain few distinct tokens
- Topic distribution generator:
  - Generates topic distributions for documents by means of a Dirichlet distribution with concentration parameter α
  - Large α ➔ probabilities are concentrated in a few topics ➔ each document will contain few topics

LDA

- The generative process for LDA corresponds to the following joint distribution of the hidden and observed variables

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) =$$

$$= \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \left(\prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n}) \right)$$

- Goal: computing the conditional distribution of the topic structure given the observed documents

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}|w_{1:D}) = \frac{p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{p(w_{1:D})}$$

LDA

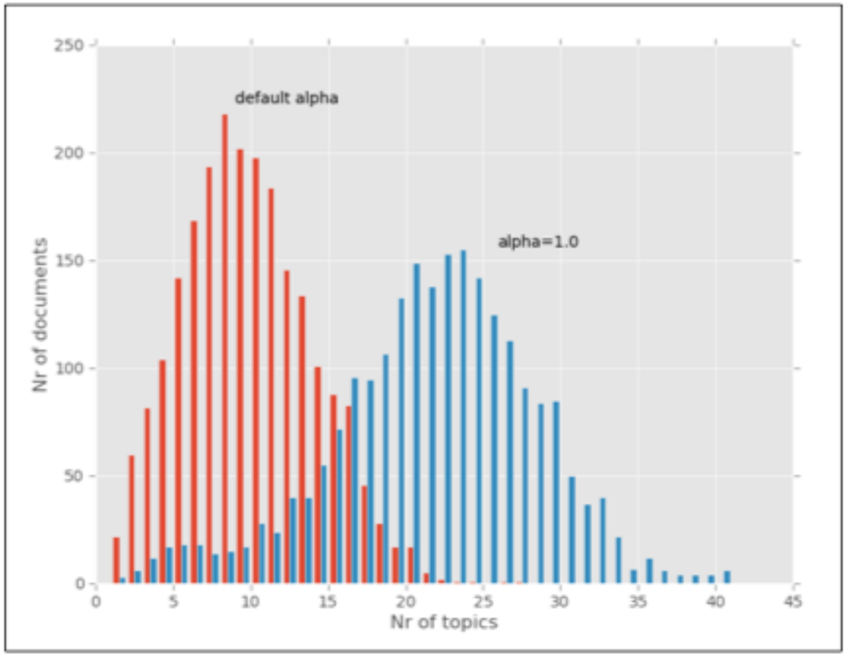
- The denominator is the probability of seeing the observed corpus under any topic model.
- It can be computed by summing the joint distribution over every possible hidden topic structure.
- The number of possible topic structures is exponentially large; this sum is intractable to compute.
- Two LDA families
  - Sampling-based algorithms
  - Variational algorithms

LDA in Gensim

- Steps:

```
# Create an LDA transformation
lda = gensim.models.LdaModel(corpus_bow,
    id2word=dictionary, alpha='auto', num_topics=20)
```

- It automatically updates the alpha value
- Bigger values for alpha will result in more topics per document



LDA in Gensim

```
>> # Analize topics
>> # (the top 5 words associated with 5 topics)
>> lda.print_topics(topics=5, topn=5)

['0.047*link + 0.027*ui + 0.018*main + 0.017*level +
0.016*locale',
 '0.107*tap + 0.047*popup + 0.045*appears +
0.031*request + 0.029*tab',
 '0.120*play + 0.096*ics + 0.084*music + 0.049*bug +
0.030*android',
 '0.106*device + 0.078*google + 0.060*talk +
0.057*voice + 0.044*icon’,
'0.191*screen + 0.055*button + 0.034*change +
0.032*page + 0.032*lock’]
```

- Weights indicate the relevance of a word for a given topic

LDA in Gensim

- More things to do....

```
# training document representation
for doc in lda:
    print(doc)

# get topic probability distribution for a document
print(lda[doc_bow])

# update the LDA model with additional documents
lda.update(corpus2)
print(lda[doc_bow])
```