

# LECTURE 6 PART 1:

# n-GRAMS

n-grams for clustering

# N-gram mixture models for clustering

- We have previously seen how mixtures of Gaussian distributions can be used to model groups of texts
- More generally, a mixture model can use any type of distribution to model the documents in each cluster.  
**N-grams can be used as the probability distribution within a cluster.**

# N-gram mixture models for clustering

- Basic idea of the EM algorithm:

- **E-step:** compute weights  $\kappa_{ik} = p(k|w^{(i)}) = \frac{p(k) p(w^{(i)}|k)}{\sum_{k'=1}^K p(k') p(w^{(i)}|k')}$   
 where  $p(w^{(i)}|\theta_k)$  is the n-gram probability for document i using the n-gram probabilities in cluster k.

- **M-step:** for each cluster k, use equations on the previous slides to optimize the N-gram parameters, but multiply transition counts from document i by  $\kappa_{ik}$ .

Example for bigrams:

$$\theta_{ML,unigram|k} = \left[ \frac{\sum_{i=1}^M \kappa_{ik} n_{1|i,unigram}}{\sum_{i=1}^M \kappa_{ik}}, \dots, \frac{\sum_{i=1}^M \kappa_{ik} n_{V|i,unigram}}{\sum_{i=1}^M \kappa_{ik}} \right]$$

$$\theta_{ML,w|k} = \left[ \frac{\sum_{i=1}^M \kappa_{ik} n_{1|w,i}}{\sum_{i=1}^M \kappa_{ik}}, \dots, \frac{\sum_{i=1}^M \kappa_{ik} n_{V|w,i}}{\sum_{i=1}^M \kappa_{ik}} \right]$$

# LECTURE 6: TOPIC MODELS



# Text corpora



- One of Finland's most popular message forums

• 20 years of conversations  
2001-2020

- 2434 discussion areas
- Over 5M discussion threads,  
16M user names

Screenshot of the Suomi24 discussion forum interface:

- Header:** SUOMI24 Etusivu Keskustelu Treffit Posti Chat Alennuskoodit KIRJAUDU
- Page Title:** keskustelu24
- Breadcrumbs:** Keskustelu24 □ Muoti ja kauneus □ Miesten muoti □ Farkkuhaalarit
- Left Sidebar (Aihealueet):**
  - Etsi aihealueelta
  - Kalikki aihealueet
  - Muoti ja kauneus
  - Alusvaatteet
  - Hiusket
  - Ihokarvat
  - Ihonhoito
  - Isokokoisten muoti ja pukeutuminen
  - Juhlavärit
  - Kauneudenhoito
  - Kellot
  - Kengät
  - Korut
  - Kynnet
  - Luontaisheidot ja kypylät
  - Lävistykset
  - Meikkauus
  - Miesten ihonholto
  - Miesten muoti
  - Naisen muoti
  - Pienikokoisen muoti ja pukeutuminen
  - Sellullitti
  - Tatuoinnit
  - Tuoksut
  - Tyylli
- Content Area:**
  - Post by Haalarirakkous:** Tuo hauska ja ihana vaate tai joidenkkin mielestä vihattu vaate eli farkkulaappuhaalarit. Nykyisin enää näkee harvojen näistä ja varsinkaan miesten pääillä farkkuhaalariteita. Itse olen n. 30 v. nainen ja pidän joskus pääilläni farkkuhaalariteita. Nyt onnistuin bongamaan miehen pääillä siniset farkkuhaalarit lauantaina 19.7.2014 Sijian Symphony laivalla Tukholman risteilyllä ja kuin sattumaa niin samoinen miehen huomasin tiistaina illalla 22.7.2014 Helsingissä Linnanmäellä farkkuhaalariteita kävelemessä.
  - Post by asdsda:** Täytty kyllä todeta, että ihastuin tähän mieheen farkkuhaalarareissaan. Hän näyttää aikas sötöltä ja hauksalta mieheitä. Ehkä hän asuu kenties pääkaupunkiseudulla. Jos tämä mies tunnistaa itsensä näistä paloista, minä voisi laittaa tälle palstalle viestitä, jos näksimme joisikin jättietyt farkkuhaalarit.
  - Post by epaselvää:** itse mies ja käytän farkkuhaalarita. on shortsihalaareita ja ihan pitkälahkeisia farkkuhaalarita.
  - Post by Haalarirakkillylä:** Enpä ole bongamaasi mies mutta vaatekaipani löytyy neljällä lappuhaalarit :)
- Bottom Navigation:** KIRJOITA VASTAUS, Jaa, Ilmanna, KESKUSTELE AIHEESTA, Luetumpia, KONTAKTI, ADON NEWS, Liittyykkääntä.

# Text corpora

SUOMI Etusivu Keskustelu Treffit Posti Chat Alennuskoodit

Suom24 yrityksille Opetus KIRJAUDU

# keskustelu<sup>24</sup>

Etsi keskustelusta

Aihaleheet

Etsi aihaleuttaa

Kalikki aihaleheet

Muoti ja kauneus

Alusvaatteet

Hiukset

Ihokarvat

Ihonhoito

Isokokoisten muoti ja pukeutuminen

Juhlapuvut

Kauneudenhoito

Kellot

Kengät

Korut

Kynnet

Luontolahdot ja kylväyt

Lävistysket

Meilkkaus

Miesten ihonhoito

Miesten muoti

Naisien muoti

Pienikokoisten muoti ja pukeutuminen

Sellullitti

Tatuointi

Tuoksut

Tyylit

**keskustelu24** □ Muoti ja kauneus □ Miesten muoti □ Farkkuhaalarit

## Farkkuhaalarit

 Haalarirakkaus  
23.7.2014 18:24

Tuo hauska ja ihana vaate tai joidenkkin mielestä vähittä vaate eli farkkulappuhaalarit. Nykyisin enää näkee harvojen naisien ja varsinkaan miesten pääillä farkkuhaalariteita. Itse olen n. 30 v. nainen ja pidän jousku pääillä farkkuhaalariteita. Nyt onnistuin bongamaan mielehen pääillä siniset farkkuhaalarit lauantaina 19.7.2014 Siljan Symphonyn laivalla Tukholman risteilyllä ja kui satumaa niin samaisen miehen huomasin tiistaina illalla 22.7.2014 Helsingissä Linnanpellällä farkkuhaalarieissa kävelémässä.

Täytty kyllä todeta, että ihastuin tähän mieheen farkkuhaalarereissaan. Hän näytti aikas sööttiltä ja hauskalta mieheltä. Ehkä hän asuu kenties pääkaupunkiseudulla. Jos tämä mies tunnistaa itsensä näistä paikoista, niin voisi laittaa tälle palstalle viestä, jos näkisimme toisemme ja tiedysti farkkuhaalarieissa.

Toivoisin näkävän enemmänkin samanhenkilöä ihmisiä sekä naisia että miehiä farkkuhaalarit pääillä ja samalla pitäisiin kokoonantaa johonkin farkkuhaalarereissa. Farkkuhaalarit ovat kivat ja rennot pitää pääillä sekä ihantat miestenkin pääillä. Mitä mieltä muut olette farkkuhaalarieista?

Laittakaa kommenttia, niin voitaisiin keskustella farkkuhaalaraleista.

Jaa Ilmianna

**46 Vastausta**

 asdada  
3.8.2014 1:09

itse mies ja käytän farkkuhaalariteita. on shortsiahaalariteita ja ihan pitkälähkeisia farkkuhaalariteita.

Jaa Ilmianna

 epäselvä  
4.8.2014 15:52

Monesko farkkuhaalariketu tämä on?

Kommentoi Jaa Ilmianna

**1 VASTAUS:**

 Pitkät houtut  
4.8.2014 17:57

Ei pysy enää laskuissa mukana, mutta farkkuhaalarit näyttäisi olevan suosittu aih. Farkkuhaalarit on jes housut.

Kommentoi lainaten Jaa Ilmianna

 Haalaritykällä  
8.8.2014 23:17

Enpäs ole bongaamasi mies mutta vaatekaipistani löytyy neljät lappuhaalarit :) Vaikka eivät ole kovin muodissa nykyisin, niin kyllä ne yllä näkyvät. Milloin missäkin, Ostarella, ystävien luona vieraillessa, jousku jopa baarissa. Jonkinkinoinen fetissi noinhan, siksiipä innoissaan näistä :)

Kommentoi Jaa Ilmianna

**3 VASTAUSTA:**

 Lappuoppsy  
10.8.2014 23:15

Useammat farkkulappuhaalarit minulta löytyy ja vaimolla on muutamat myös. Ihan huippu houstit mielellä. Sopii myös naisille, jos ovat naisiltaan mallia.

Kommentoi lainaten Jaa Ilmianna

 henkiselman  
11.8.2014 13:13

Lappuoppsy kirjoitti:  
Useammat farkkulappuhaalarit minulta löytyy ja vaimolla on muutamat myös. ihan huippu houstit mi...

**Keskustele**

KESKUSTELE AIHEESTA

**Luetuimpia**

mistä löytyää skinny -malliset farkk...  
mistä löytyää skinny -malliset farkk...

sukkahousut ollis perannut kun hi...  
Sukkahousut ollis perannut mukavat...

Mies ja kolmostila alkava kengän...  
Onko kohtaloitoverita? Miehiä, joide...

Duffellakki suomesta  
Olkoon kukaan nähty jossakin liikee...

Taisustuu!  
Mulla pikkut kireät nahkahousut, nap...

**E-kontakti.fi**

  
23.v Pohjanmaa

  
31.v Uusimaa

  
26.v Uusimaa

  
40.v Uusimaa

  
49.v Uusimaa

  
38.v Päijät-Häme

**LÖYDÄ SEURAA  
TOSITÄRKEÄKSELLÄ!**

Haen naista: 25-39 v. 40-54 v. 55+ v.  
Haen miestä: 25-39 v. 40-54 v. 55+ v.

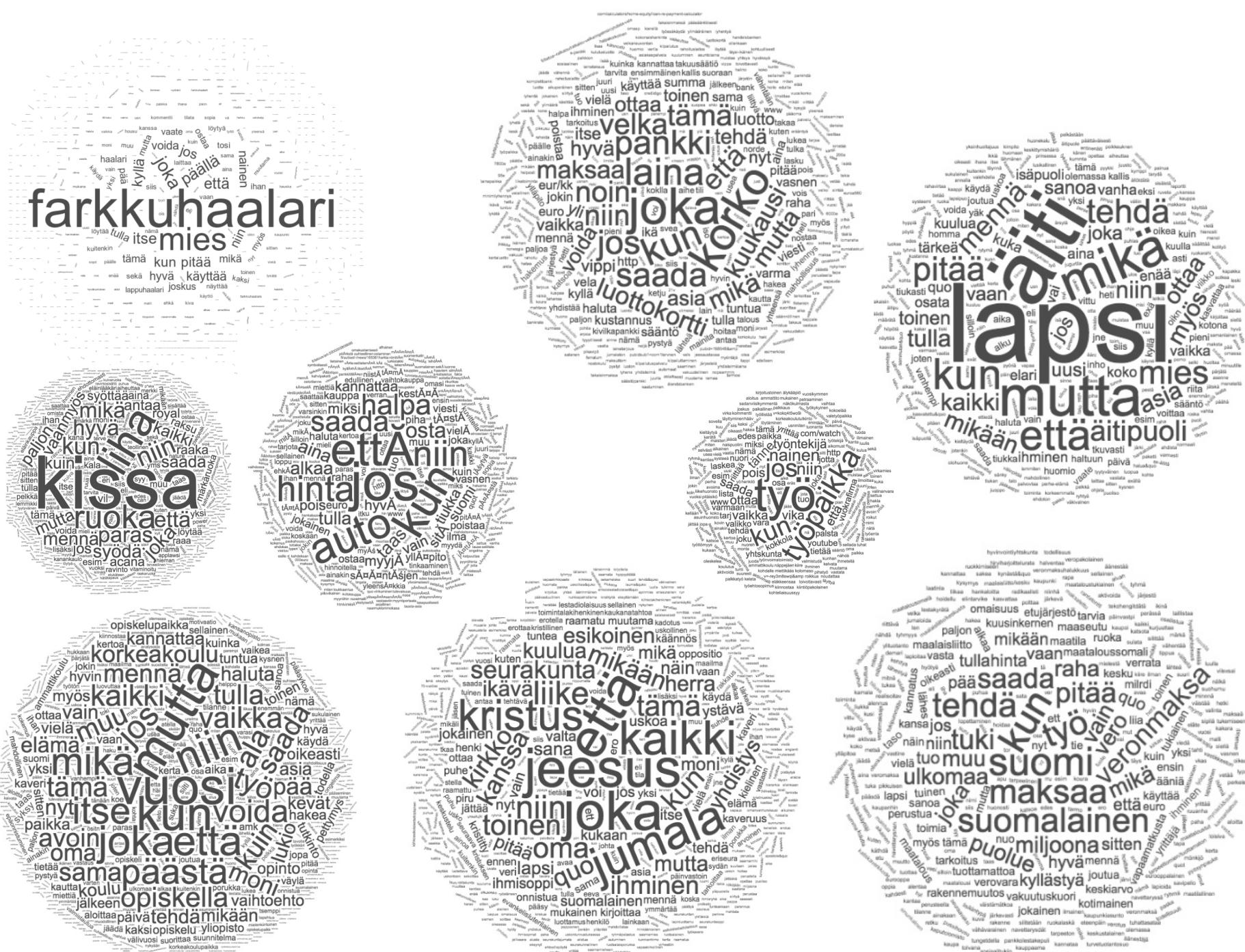
**Nyt voi  
yrityksesi  
näkyä tassä!**

**Aller media** **Lisätieto Klikka tästä >**

**ADON** **HOVS**

**Klikkaa tästä**

# Text corpora



# Topic modeling: intuitive idea



# Topic modeling: intuitive idea



# Topic modeling: intuitive idea



# Topic modeling: intuitive idea



?

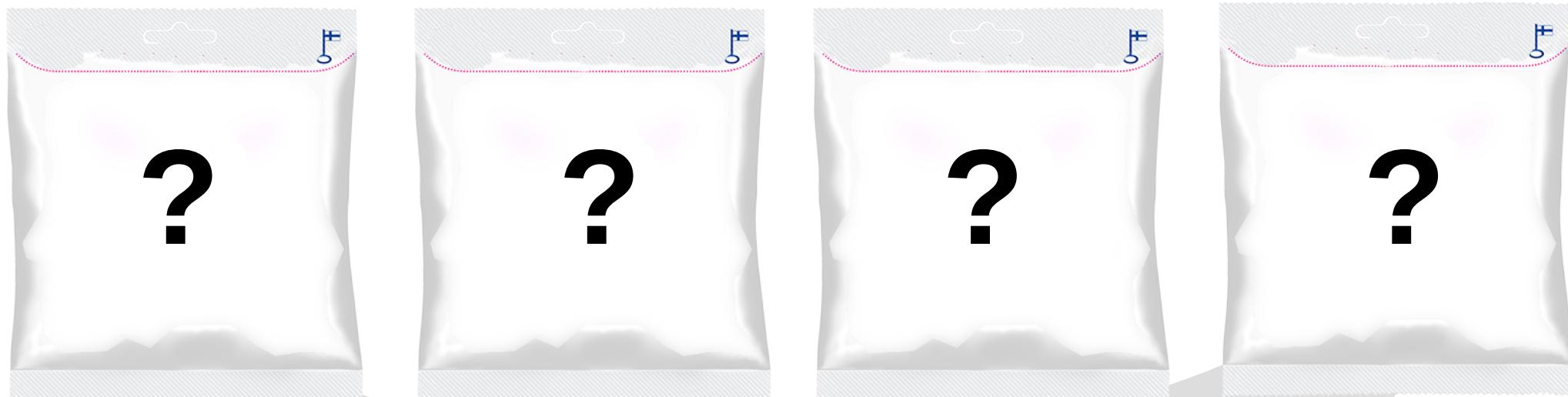
?

?

?



# Topic modeling: intuitive idea



?

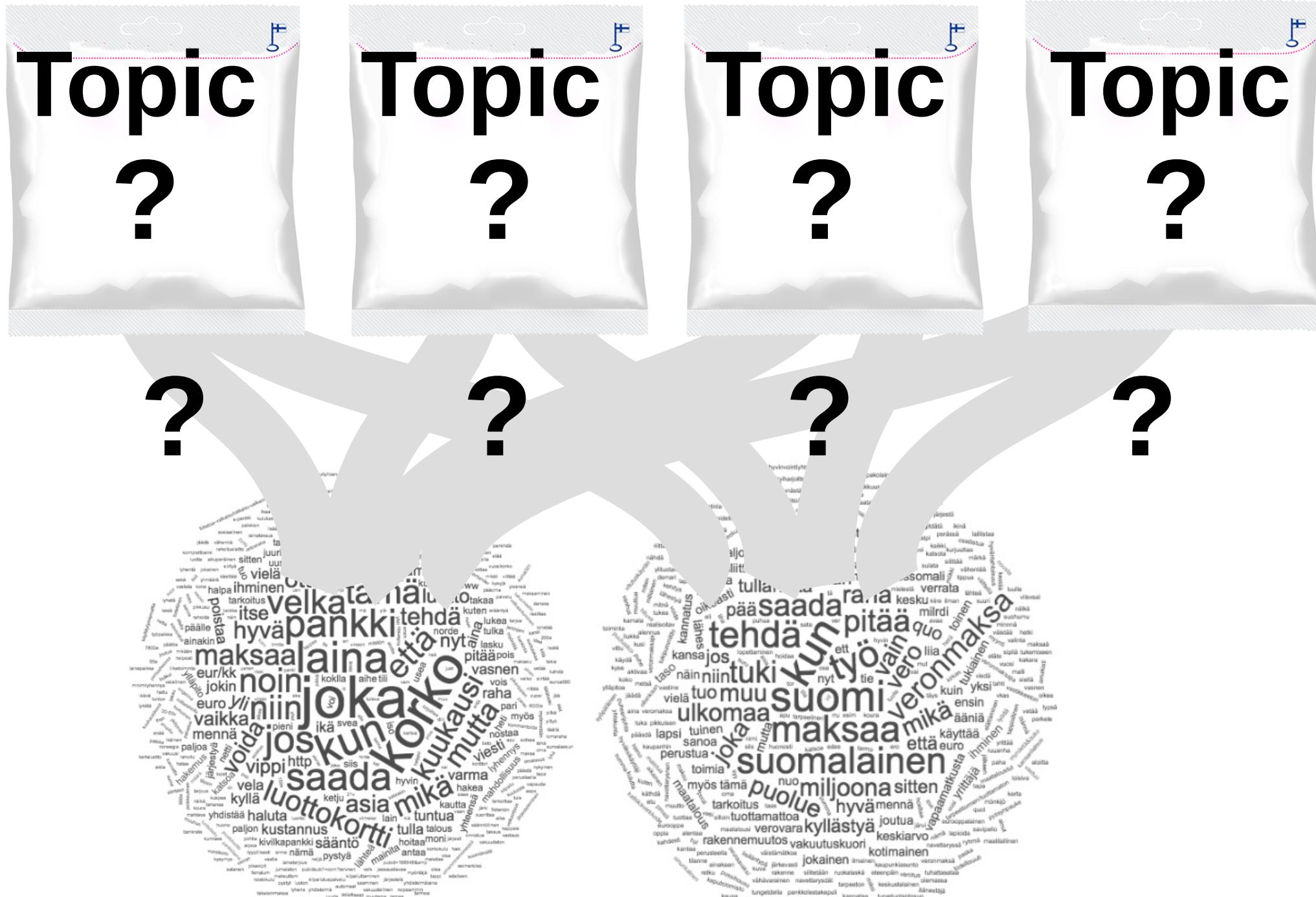
?

?

?



# Topic modeling: intuitive idea



# Latent Semantic Analysis

- **Latent semantic analysis** (LSA; also called latent semantic indexing) is a statistical method to break a term frequency matrix into underlying factors.
- It is a vector-space based linear dimensionality reduction method for term-frequency vectors. It creates a low-rank approximation of a term-frequency matrix.
- Procedure: For M documents and V terms in the vocabulary,
  - Create a term-frequency matrix  $\mathbf{X}$  (M rows, V columns)
  - Compute a singular value decomposition of the matrix:

$$\mathbf{X} = \mathbf{U}_{\text{Left}} \mathbf{D} \mathbf{U}_{\text{Right}}^T$$

$\mathbf{U}_{\text{Left}}$  is a M\*K matrix of left eigenvectors, K=min(M,V)  
 $\mathbf{D}$  is a K\*K diagonal matrix of singular values  
 $\mathbf{U}_{\text{Right}}$  is a V\*K matrix of right eigenvectors

-Take the largest few (e.g. k=20 largest) diagonal elements of D, take the corresponding columns of  $\mathbf{U}_{\text{Left}}$  and  $\mathbf{U}_{\text{Right}}$ : call them  $\hat{\mathbf{U}}_{\text{Left}}, \hat{\mathbf{D}}, \hat{\mathbf{U}}_{\text{Right}}$  (they are sized M\*k, k\*k and V\*k respectively)

# Latent Semantic Analysis

- Then  $\hat{\mathbf{X}} = \hat{\mathbf{U}}_{\text{Left}} \hat{\mathbf{D}} \hat{\mathbf{U}}_{\text{Right}}^T$  approximates the original matrix
- Each document  $i$  (row  $i$  of  $\mathbf{X}$ ) is approximated as
$$\mathbf{X}[i, :] \doteq \hat{\mathbf{U}}_{\text{Left}}[i, :] \hat{\mathbf{D}} \hat{\mathbf{U}}_{\text{Right}}^T = \sum_{j=1}^k \hat{\mathbf{U}}_{\text{Left}}[i, j] \hat{\mathbf{D}}[j, j] \hat{\mathbf{U}}_{\text{Right}}[:, j]^T$$
- This approximation represents each document as a weighted combination of the columns of  $\hat{\mathbf{U}}_{\text{Right}}$ , so that
  - Each column of the  $M \times k$  matrix  $\hat{\mathbf{U}}_{\text{Left}}$  is a low-dimensional representation of a document
  - Each column of the  $V \times k$  matrix  $\hat{\mathbf{U}}_{\text{Right}}$  is a word-loading vector that represents a direction of variation in the data: which word counts tend to vary together
- LSA is the same as **principal component analysis** (PCA) on a covariance matrix of terms, computed from  $\mathbf{X}$ , except that PCA subtracts means, LSA does not
- LSA can also be applied to e.g. TF-IDF matrices



A baseball game is in progress on a green field with a dirt infield. A batter is at home plate, ready to swing. An umpire stands behind him, and a catcher is crouched down. The field has white chalked lines and a large blue and yellow logo in the grass. In the background, there are trees, stadium lights, and a scoreboard. The scoreboard displays "UCLA FRIENDS" and "EASTON".

**Take Me Out To The  
Ball Game**

# Latent Semantic Analysis

- Example code in Python:

```
## Try LSA on a very small data set
# Let's take the top-500 TF-IDF features,
# and 1000 documents of 20 Newsgroups
# (the rec.sport.baseball newsgroup)

dimensiontotals=numpy.squeeze( \
    numpy.array(numpy.sum(tfidfmatrix, axis=0)))
highesttotals=numpy.argsort(-1*dimensiontotals)
Xsmall=tfidfmatrix[:,highesttotals[0:500]]
Xsmall=Xsmall[9000:10000,:].todense()

# Compute 10 factors from LSA
n_low_dimensions=10
Uleft,D,UrightT=scipy.sparse.linalg.svds(\n    Xsmall,k=n_low_dimensions)
```

# Latent Semantic Analysis

- Example code in Python:

```
# Examine the singular values
print(D)
Out: [ 76.78596317  82.07492002  91.14531964  98.92853817 113.17000393
      119.25390774 143.97132142 154.20292869 196.08939846 238.9600725 ]
# Examine a factor (here the one with largest singular value)
print(UrightT[9,:])
Out: [ 2.76185154e-18  3.33910697e-18 -1.36216218e-18 -3.12435117e-04
      -2.20575924e-18 -3.80957762e-05 -4.17970680e-18  1.03556854e-18
      -1.33681293e-18  8.83776226e-19  4.67368313e-18  2.91952229e-18
      4.23182966e-18 -1.23820240e-18  7.28796702e-19 -1.21670041e-18
      -4.82451750e-05 -1.02934627e-05 -2.71219158e-18 -3.78173411e-05
      -3.96785618e-18  2.76640163e-19 -1.03155284e-05 -3.21182175e-08
      -1.76191160e-18  1.71208356e-18 -1.34796924e-06 -2.08766247e-18
# 20 words with largest absolute weights in the factor
topweights_indices=numpy.argsort(-1*numpy.abs(UrightT[9,:]))
print(remainingvocabulary[highesttotals[topweights_indices[0:20]]])
['gant' 'plate' 'dale' 'cub' 'inning' 'earlier' 'dispute' 'randy'
 'subscribe' 'struggle' 'ridiculous' 'glad' 'ab' 'ml' 'defensive' 'stl'
 'impression' 'bond' 'wayne' 'complaint']
# Same for the next highest factor
topweights_indices=numpy.argsort(-1*numpy.abs(UrightT[8,:]))
print(remainingvocabulary[highesttotals[topweights_indices[0:20]]])
['cub' 'inning' 'sox' 'idle' 'baltimore' 'era' 'stl' 'friday' 'houston'
 'saturday' 'bond' 'florida' 'minnesota' 'august' 'gant' 'ab' 'angel'
 'weekend' 'wilson' 'aaron']
```

# Probabilistic Latent Semantic Analysis

- **Probabilistic latent semantic analysis** (PLSA; also called probabilistic latent semantic indexing) is a probabilistic model of co-occurrences of words and documents.
  - References:
    - Thomas Hofmann. **Probabilistic Latent Semantic Indexing**. In *Proceedings of the International Conference on Research and Development in Information Retrieval*, 1999.
    - Thomas Hofmann. **Unsupervised Learning by Probabilistic Latent Semantic Analysis**. *Machine Learning*, 42, 177-196, 2001
  - Consider a collection with a set of  $M$  documents  $d$  and a vocabulary with a set of  $V$  words  $w$ . PLSA models the **co-occurrence** of a particular word in a particular document

# Probabilistic Latent Semantic Analysis

- The joint probability of  $d$  and  $w$  is modeled given underlying latent classes  $t$ : the probabilities of  $d$  and  $w$  are assumed to become **independent** given the class. "If I know what I'm talking about, where I am has no effect on what I say!"

$$p(w, d) = \sum_{t=1}^T p(t) p(d|t) p(w|t) = \sum_{t=1}^T \pi_t \psi_{d|t} \theta_{w|t}$$

- This can be rewritten as

$$p(w, d) = p(d) p(w|d) = p(d) \sum_{t=1}^T p(t|d) p(w|t) = \pi_d \sum_{t=1}^T \psi_{t|d} \theta_{w|t}$$

- Here  $\pi_d = p(d)$ ,  $\psi_{t|d} = p(t|d)$ , and  $\theta_{w|t} = p(w|t)$  are probabilities of multinomial distributions which are parameters of the PLSA model.

# Probabilistic Latent Semantic Analysis

- In a set of  $M$  documents, suppose document  $d_i$  has  $N_i$  word occurrences. The PLSA log-likelihood is:

$$\sum_{i=1}^M \sum_{j=1}^{N_i} \log p(w_j, d_i) = \sum_{i=1}^M \sum_{j=1}^{N_i} \log (\pi_i \sum_{t=1}^T \psi_{t|d} \theta_{w|t})$$

- Another way to represent the same thing: if we count how many co-occurrences  $n_{v,i}$  each vocabulary word  $v$  has in document  $i$ , the log-likelihood can be written as

$$\sum_{i=1}^M \sum_{v=1}^V n_{v,i} \log (\pi_i \sum_{t=1}^T \psi_{t|d} \theta_{w|t})$$

- Each word occurrence comes from one topic  $t$ , which can be seen as a latent variable. PLSA can be fitted to a set of documents by maximum likelihood using the **Expectation-Maximization algorithm**.

# Probabilistic Latent Semantic Analysis

- **E-step:** for each pair  $v, d_i$  compute the conditional topic distribution

$$\begin{aligned} \kappa_{t|v,i} &= p(t|v, d_i) = \frac{p(v, d_i, t)}{p(v, d_i)} = \frac{p(v|t)p(t|d_i)p(d_i)}{p(v, d_i)} \\ &= \frac{\theta_{v|t}\psi_{t|d_i}\pi_{d_i}}{\sum_{t'=1}^T \theta_{v|t'}\psi_{t'|d_i}\pi_{d_i}} = \frac{\theta_{v|t}\psi_{t|d_i}}{\sum_{t'=1}^T \theta_{v|t'}\psi_{t'|d_i}} \end{aligned}$$

- **M-step:** optimize the parameters as

$$\pi_{n_i} = \frac{\sum_{v=1}^V n_{v,i}}{\sum_{i'=1}^M \sum_{v=1}^V n_{v,i'}}$$

$$\psi_{t|d_i} = \frac{\sum_{v=1}^V n_{v,i} \kappa_{t|v,i}}{\sum_{v=1}^V n_{v,i}}$$

$$\theta_{v|t} = \frac{\sum_{i=1}^M n_{v,i} \kappa_{t|v,i}}{\sum_{i=1}^M \sum_{v'=1}^V n_{v',i} \kappa_{t|v',i}}$$

These equations are simple to implement: no need for any complicated mathematical operations, just simple sums. The  $\pi_d = p(d)$  only need to be computed once.

# Probabilistic Latent Semantic Analysis

- PLSA code in Python (there are also a number of existing implementations available online):

```

import numpy, numpy.matlib, scipy, scipy.stats
def plsa(document_to_word_matrix, n_topics, n_iterations):
    n_docs=numpy.shape(document_to_word_matrix)[0]                                # Number of documents and vocabulary words
    n_vocab=numpy.shape(document_to_word_matrix)[1]
    theta = scipy.stats.uniform.rvs(size=(n_vocab,n_topics)) # Prob of words per topic: random init
    theta = theta/numpy.matlib.repmat(numpy.sum(theta, axis=0), n_vocab, 1)
    psi = scipy.stats.uniform.rvs(size=(n_topics,n_docs))      # Probs topics per document: random init
    psi = psi/numpy.matlib.repmat(numpy.sum(psi, axis=0), n_topics, 1)
    n_words_in_docs = numpy.squeeze(numpy.array(\                                         # Numbers of words in documents: computed once
        numpy.sum(document_to_word_matrix, axis=1)))
    n_totalwords = numpy.sum(n_words_in_docs)                                     # Total number of words: computed once
    pi = n_words_in_docs/n_totalwords                                           # Document probs: computed once
    for myiter in range(n_iterations):                                         # Perform Expectation-Maximization iterations
        # ===Perform E-step=====
        doc_word_to_topics = []                                              # Compute theta_{v|t}psi_{t|d}/sum_t' theta_{v|t'}psi_{t'|d}
        doc_word_to_topic_sum = numpy.zeros((n_docs,n_vocab))
        for t in range(n_topics):
            doc_word_to_topict = \
                numpy.matlib.repmat(theta[:,t], n_docs, 1) * \
                numpy.matlib.repmat(psi[t,:], n_vocab, 1).T
            myepsilon=1e-14           # Add a positive number to avoid divisions by zero
            doc_word_to_topict += myepsilon
            doc_word_to_topics.append(doc_word_to_topict)
            doc_word_to_topic_sum += doc_word_to_topict
        for t in range(n_topics):
            doc_word_to_topics[t] /= doc_word_to_topic_sum
        # =====Perform M-step=====
        # Add a small number to word counts to avoid divisions by zero
        for t in range(n_topics):                                              # Compute document-to-topic probabilities.
            psi[t,:] = numpy.squeeze(numpy.array(numpy.sum( \
                numpy.multiply(document_to_word_matrix+myepsilon, doc_word_to_topics[t]), axis=1)))
        psi /= numpy.matlib.repmat(numpy.sum(psi, axis=0), n_topics, 1)
        for t in range(n_topics):                                              # Compute topic-to-word probabilities
            theta[:,t]= numpy.squeeze(numpy.array(numpy.sum( \
                numpy.multiply(document_to_word_matrix, doc_word_to_topics[t]), axis=0).T))
        theta /= numpy.matlib.repmat(numpy.sum(theta, axis=0), n_vocab, 1)
    return(pi,psi,theta)

```

# Probabilistic Latent Semantic Analysis

- Example code in Python:

```
#% Try PLSA on a very small data set
# Let's take the same 500 features as in LSI,
# and the same 1000 baseball documents but use
# the term-frequency (TF) values for the features
dimensiontotals=numpy.squeeze( \
    numpy.array(numpy.sum(tfidfmatrix, axis=0)))
highesttotals=numpy.argsort(-1*dimensiontotals)
Xsmall=tfmatrix[:,highesttotals[0:500]]
Xsmall=Xsmall[9000:10000,:].todense()

# Run PLSA
n_topics=10
n_iterations=200
pi,psi,theta=plsa(Xsmall, n_topics, n_iterations)
```

# Probabilistic Latent Semantic Analysis

- Example code in Python:

```
# Examine the factor probabilities p(t) = sum_d p(t|d) p(d)
print(numpy.sum(psi*numpy.matlib.repmat(pi,n_topics,1),axis=1))
[0.11280546 0.11582869 0.09660424 0.10031745 0.11351756 0.09626955
 0.10611472 0.08149722 0.09108973 0.0859554]

# Examine a factor (here the one with largest probability, factor 1)
print(theta[:,1])
[0.0000000e+00 0.0000000e+00 0.0000000e+00 7.74240115e-15
 0.0000000e+00 4.15656746e-14 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 1.40467285e-14 9.66468649e-15 0.0000000e+00 8.85596500e-15
 0.0000000e+00 0.0000000e+00 8.66068367e-15 9.80049211e-15

# Words with largest absolute weights in the factor
topweights_indices=numpy.argsort(-1*numpy.abs(theta[:,1]))
print(remainingvocabulary[highesttotals[topweights_indices[0:20]]])
['inning' 'idle' 'minnesota' 'florida' 'fourth' 'saturday' 'eight' 'grab'
 'award' 'furthermore' 'mainly' 'effectively' 'supposedly' 'tx' 'ai'
 'combine' 'phenomenon' 'divide' 'lift' 'chief']

# Same for the next biggest factor (here factor 4)
topweights_indices=numpy.argsort(-1*numpy.abs(theta[:,4]))
print(remainingvocabulary[highesttotals[topweights_indices[0:20]]])
['bond' 'defensive' 'offensive' 'matt' 'dale' 'struggle' 'impression'
 'conservative' 'lift' 'edward' 'sp' 'math' 'lewis' 'negative' 'widely'
 'politics' 'progress' 'significantly' 'terrible' 'related']
```

# Probabilistic Latent Semantic Analysis

- In PLSA, content of an individual document is represented by the distribution

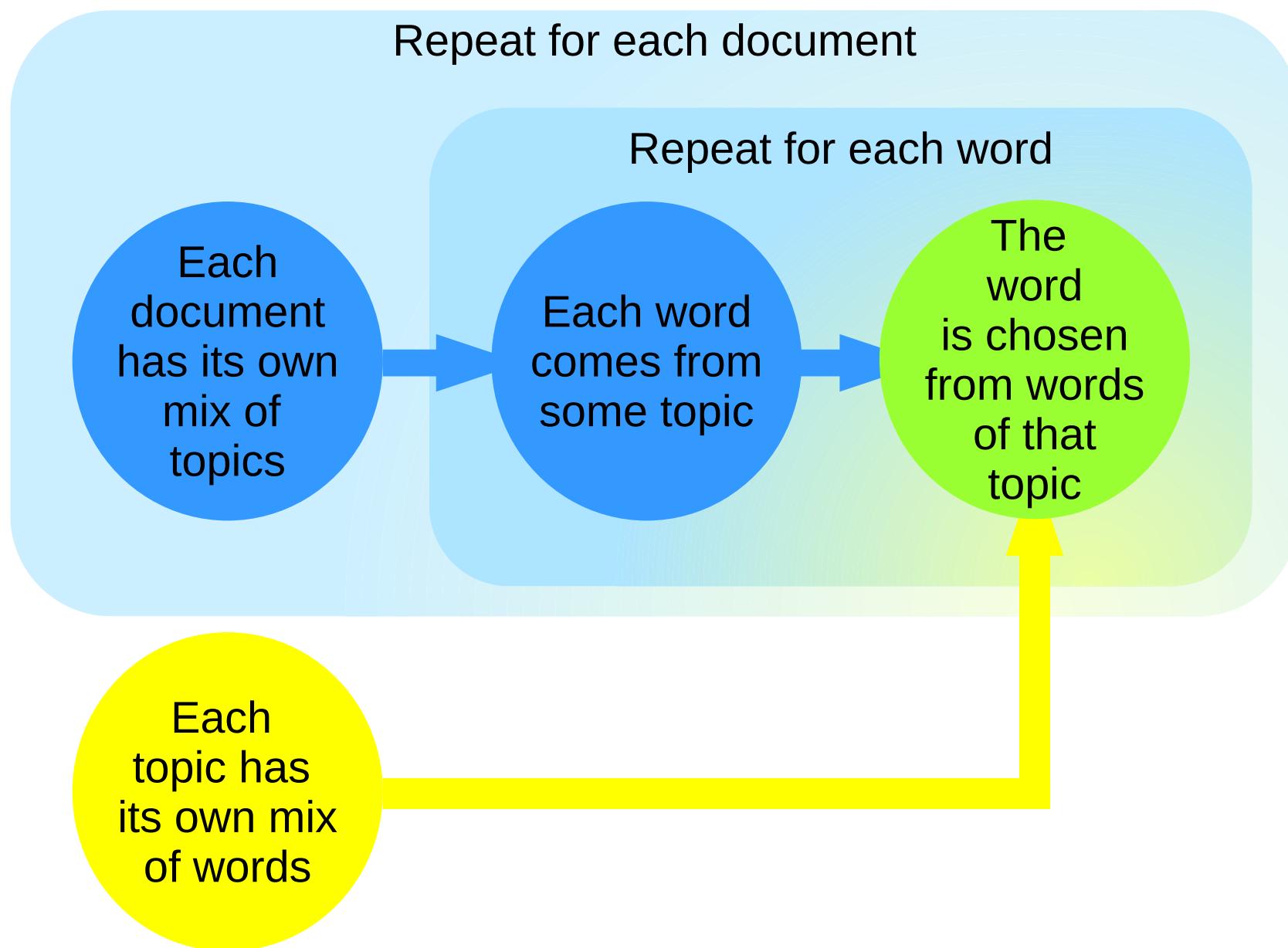
$$p(w|d) = \frac{p(w,d)}{p(d)} = \frac{p(d) \sum_{t=1}^T p(w|t)p(t|d)}{p(d)} = \sum_{t=1}^T \psi_{t|d} \theta_{w|t}$$

- Problem: PLSI is only a generative model of the observed set of documents - not of new documents!
  - PLSI defines  $\psi_{t|d} = p(t|d)$  only over existing documents d, so one cannot derive what topics would exist in a new document

# Latent Dirichlet Allocation

- Latent Dirichlet Allocation (LDA): a probabilistic model that discovers thematic topics underlying text documents, without requiring pre-existing definitions of what themes to look for.
  - Reference: David M. Blei, Andrew Y. Ng, and Michael I. Jordan. **Latent Dirichlet Allocation**. In *Proceedings of Neural Information Processing Systems 2002*.
- Topics are "building blocks" of text: each document is composed of a mixture of topics.
- This is different from mixture modeling:
  - In mixture modeling each document actually comes from only one underlying mixture component, even if it is uncertain which one.
  - In topic modeling each document really contains multiple topics at the same time.
- LDA is sometimes called a "discrete Principal Component Analysis"

# Latent Dirichlet Allocation: plate model



# Latent Dirichlet Allocation

- Probability of a word  $w$  occurring in a document  $d$  is a sum over a finite set of  $T$  topics:

$$p(w|d; \pi_d) = \sum_{t=1}^T p(t|d) p(w|t) = \sum_{t=1}^T \pi_{t|d} \theta_{w|t}$$

- Here  $\pi_{t|d} = p(t|d)$  is the probability (proportion, strength, prevalence) of topic  $t$  in document  $d$ , and  $\theta_{w|t} = p(w|t)$  is the probability of word  $w$  in topic  $t$ . The  $\pi_{t|d}$  and  $\theta_{w|t}$  are parameters of the LDA model.
- The vector of topic probabilities  $\pi_d = \{\pi_{t|d}\}$  in the document is assumed to arise from a Dirichlet prior:  $\pi_d \sim Dirichlet(\alpha)$
- The probability of the word contents of a document must then be integrated over the possible topic probabilities

# Latent Dirichlet Allocation

- Suppose document  $d_i$  has  $N_i$  word occurrences. Denote the word contents of the document by  $\mathbf{d}_i$ . The likelihood of the document contents is:

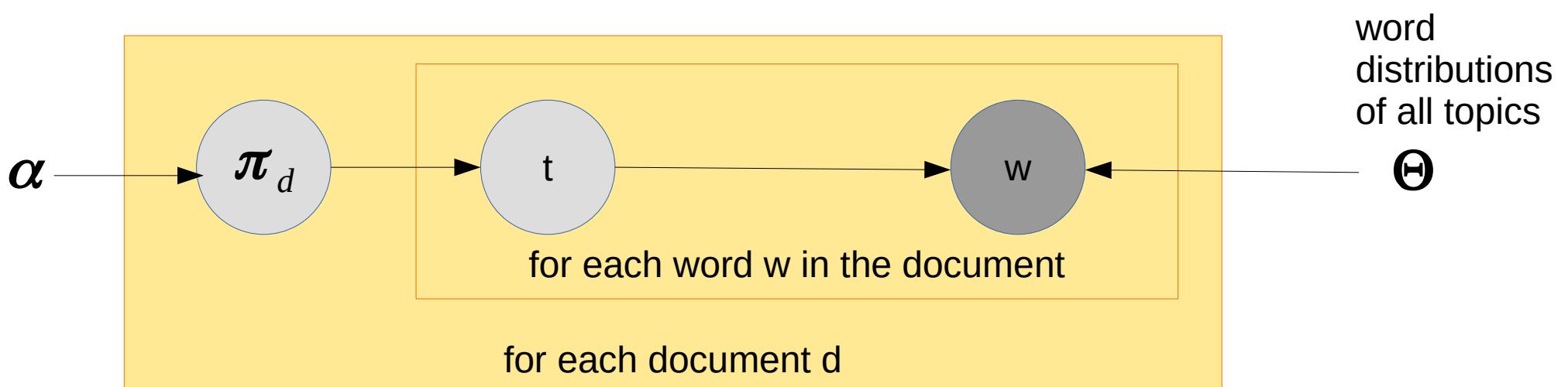
$$\begin{aligned} p(\mathbf{d}_i; \boldsymbol{\alpha}, \Theta) &= \int_{\boldsymbol{\pi}_{d_i}} p(\boldsymbol{\pi}_d | \boldsymbol{\alpha}) \left( \prod_{j=1}^{N_i} p(w_j | d_i, \boldsymbol{\pi}_{d_i}) \right) d\boldsymbol{\pi}_{d_i} \\ &= \int_{\boldsymbol{\pi}_d} p(\boldsymbol{\pi}_d | \boldsymbol{\alpha}) \left( \prod_{j=1}^{N_i} \left( \sum_{t=1}^T \pi_{t|d_i} \theta_{w_j|t} \right) \right) d\boldsymbol{\pi}_d \end{aligned}$$

and the likelihood of several documents is the product of the above terms for each:  $\prod_{i=1}^M p(\mathbf{d}_i)$

- The topic probabilities are specific to each document but the word probabilities per topic are common to all documents.

# Latent Dirichlet Allocation

- The corresponding plate model:



# Latent Dirichlet Allocation

- This function is cannot be analytically computed, and is complicated to optimize for a set of data
- Two typical optimization approaches are used: **variational approximation** and **Gibbs sampling**
- Denote by  $\mathbf{t}_d$  a vector containing the topic choice of each word in document d. Then the probability of the document content can be written as a sum over possible vectors  $\mathbf{t}_d$

$$\log p(\mathbf{d}; \boldsymbol{\alpha}, \Theta) = \log \int_{\boldsymbol{\pi}_d} \sum_{\mathbf{t}_d} p(\mathbf{w}_d | \mathbf{t}_d) p(\mathbf{t}_d | \boldsymbol{\pi}_d) p(\boldsymbol{\pi}_d | \boldsymbol{\alpha}) d \boldsymbol{\pi}_d$$

- In variational approximation the idea is to compute an approximation of the posterior distribution of LDA parameters given data. The approximate distribution  $q()$  is assumed to have a simple form, and it is optimized to be close to the real posterior

$$\begin{aligned} \log p(\mathbf{d}; \boldsymbol{\alpha}, \Theta) &= \log \int_{\boldsymbol{\pi}_d} \sum_{\mathbf{t}_d} p(\mathbf{w}_d | \mathbf{t}_d) p(\mathbf{t}_d | \boldsymbol{\pi}_d) p(\boldsymbol{\pi}_d | \boldsymbol{\alpha}) \frac{q(\boldsymbol{\pi}, \mathbf{t}_d; \gamma, \phi)}{q(\boldsymbol{\pi}, \mathbf{t}_d; \gamma, \phi)} d \boldsymbol{\pi}_d \\ &\geq E_q [\log p(\mathbf{w}_d | \mathbf{t}_d; \Theta) + \log p(\mathbf{t}_d | \boldsymbol{\pi}_d) + \log p(\boldsymbol{\pi}_d | \boldsymbol{\alpha}) - \log q(\boldsymbol{\pi}_d, \mathbf{t}_d; \gamma, \phi)] \end{aligned}$$

- The  $q()$  is chosen to be  $q(\boldsymbol{\pi}, \mathbf{t}_d; \gamma, \phi) = \text{Dirichlet}(\boldsymbol{\pi}; \gamma) \prod_{i=1}^{N_d} \text{Multinomial}(z_i; \phi_i)$

# Latent Dirichlet Allocation

- In Gibbs sampling the idea is:
  - Set  $\mathbf{t}_d$  at first be some initial vector of topic choices for each word
  - For each word  $i$ , temporarily remove that word from its topic, and choose a new topic for it: compute for all  $k=1,\dots,T$   
 $p(t_i=k|w_d, \mathbf{t}_{d,-i}, \boldsymbol{\pi}_d, \Theta)$  where  $\mathbf{t}_{d,-i}$  denotes the topic choices of all other words except  $i$  (when we are dealing with multiple documents, it would denote topic choices of all other words in all documents), and then sample a new topic choice for  $i$ .
  - Do the same for other parameters
- However, in practice people usually use collapsed Gibbs sampling for LDA (reference: Thomas L Griffiths and Mark Steyvers. **Finding scientific topics**. *Proceedings of the National academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.)
- In collapsed Gibbs the idea is to also add a Dirichlet prior for the word distributions  $\boldsymbol{\theta}_t$ , and then integrate over the distributions  $\boldsymbol{\pi}_d$  and  $\Theta$
- The result will be probabilities for resampling the topic-choice of each word, that only depend on topic-choices of the other words:  $p(t_i=k|w_d, \mathbf{t}_{d,-i})$
- This resampling is done for multiple iterations and then afterwards the topic model parameters can be computed as straightforward estimates based on the topic-assignments of words.

# Latent Dirichlet Allocation

- There are several Python implementations of LDA: we will use the **Gensim** library implementation.
- Code:

```
import gensim
# Create a dictionary from the documents
startdoc=9000      # We will use the baseball n.g.
enddoc=10000        # again, but all features
gensim_docs=mycrawled_prunedtexts[startdoc:enddoc]
gensim_dictionary=gensim.corpora.Dictionary(gensim_docs)
# Create the document-term vectors
gensim_docvectors=[]
for k in range(len(gensim_docs)):
    docvector=gensim_dictionary.doc2bow(gensim_docs[k])
    gensim_docvectors.append(docvector)
```

# Latent Dirichlet Allocation

- There are several Python implementations of LDA: we will use the **Gensim** library implementation.
- Code:

```
# Run the LDA optimization
numtopics=10
randomseed=124574527
numiters=10000
ninit=10
gensim_ldamodel=gensim.models.ldamodel.LdaModel( \
    gensim_docvectors, \
    id2word=gensim_dictionary, num_topics=numtopics, \
    iterations=numiters, random_state=randomseed)
```

# Latent Dirichlet Allocation

- There are several Python implementations of LDA: we will use the **Gensim** library implementation.
- Code:

```
# Get topic content: term-topic probabilities
gensim_termtopicprobabilities=gensim_ldamodel.get_topics()
# Get topic prevalences per document, and overall topic prevalences
# (expected amount of documents per topic)
overallstrengths=numpy.zeros((numtopics,1))
documentstrengths=numpy.zeros((len(gensim_docvectors),numtopics))
for k in range(len(gensim_docvectors)):
    topicstrengths=gensim_ldamodel.get_document_topics(\n        gensim_docvectors[k],minimum_probability=0)
    for m in range(len(topicstrengths)):
        documentstrengths[k][topicstrengths[m][0]]=topicstrengths[m][1]
        overallstrengths[topicstrengths[m][0]]=\
            overallstrengths[topicstrengths[m][0]]+topicstrengths[m][1]
```

# Latent Dirichlet Allocation

- There are several Python implementations of LDA: we will use the **Gensim** library implementation.
- Code:

```
# Show top-10 words of the top topic (number 5 here)
print(gensim_ldamodel.show_topic(5,topn=10))
[('mattingly', 0.011312583), ('sox', 0.010481743), ('idle', 0.010074499),
('williams', 0.008472339), ('rbi', 0.008317692), ('bond', 0.008116004),
('pitching', 0.006333267), ('batter', 0.00531599), ('batting', 0.004842933),
('sandberg', 0.0047366144)]
```

```
# Show top-10 words of the next strongest topic (number 4 here)
print(gensim_ldamodel.show_topic(5,topn=10))
[('winfield', 0.010770354), ('inning', 0.009858241), ('tiger', 0.00792999),
('clemens', 0.006710715), ('sox', 0.0063109742), ('boggs', 0.0054057315),
('hr', 0.005192798), ('era', 0.0049536), ('rbi', 0.0047922763), ('pitching',
0.004459158)]
```