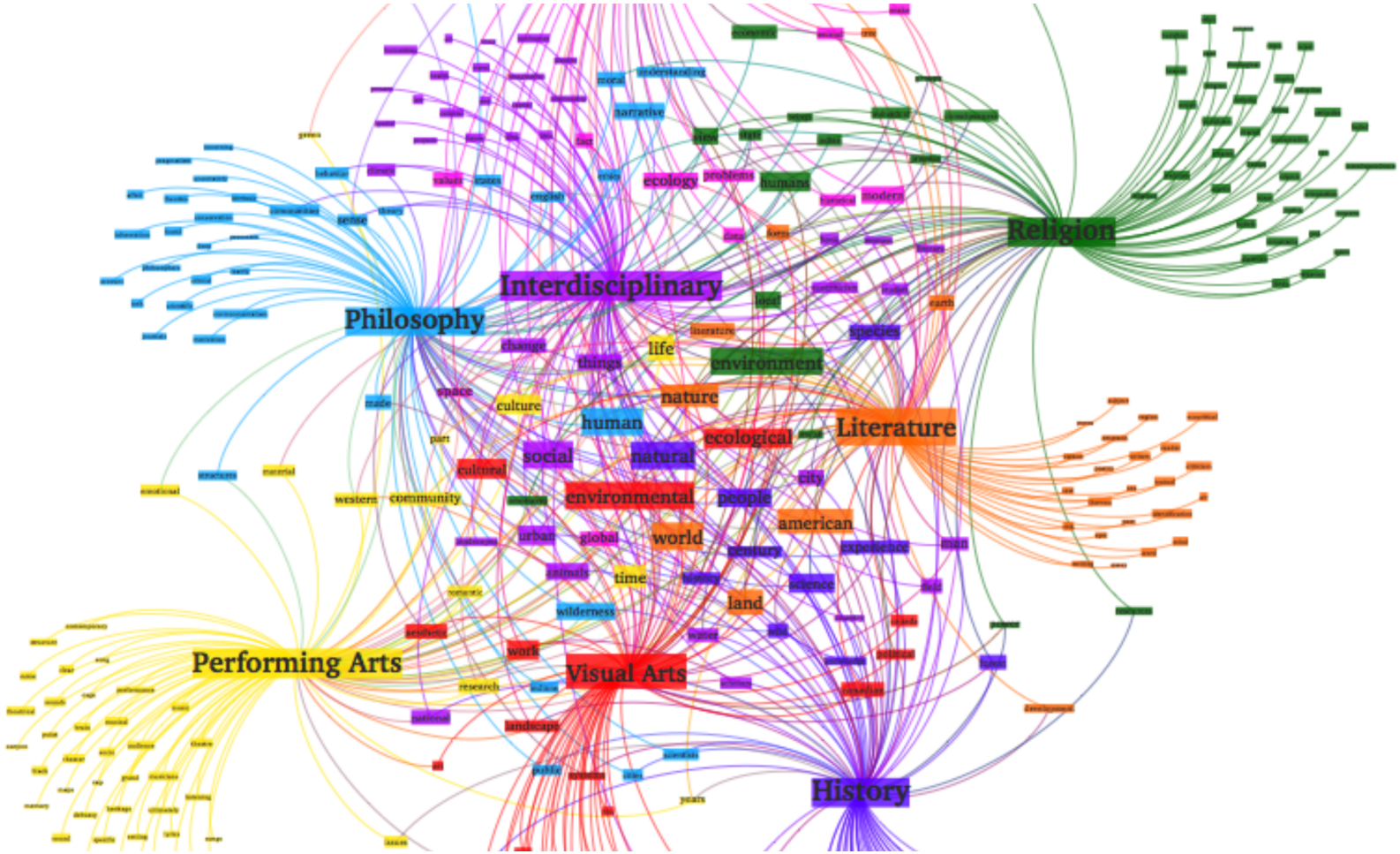
Topic Modelling

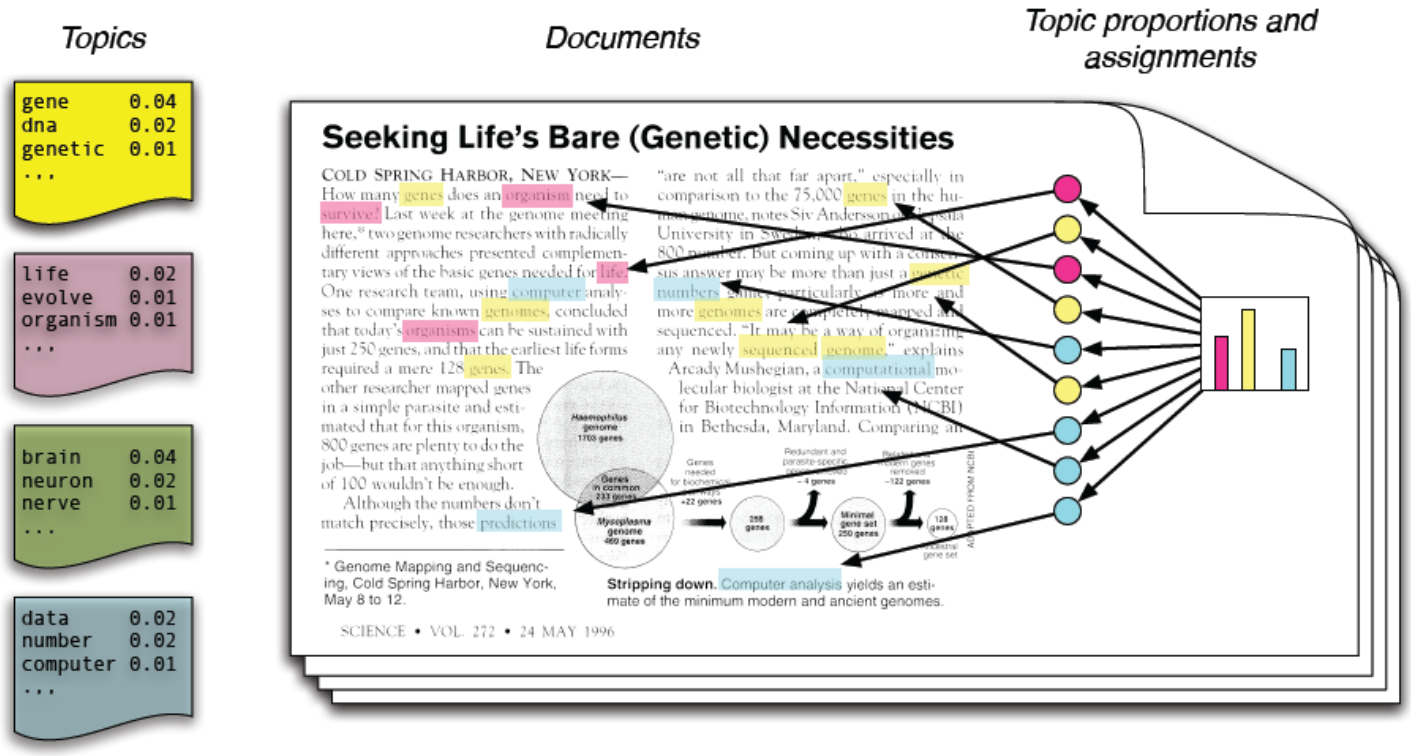


Topic modelling

- Subfield of Natural Language Understanding
- Learning, recognizing, extracting *topics* across a collection of documents
  - LSA: Latent Semantic Analysis
  - pLSA: Probabilistic LSA
  - LDA: Latent Dirichlet Allocation
    - Bayesian version of LSA
  - lda2vec: deep learning LDA
    - word2vec: LSE vector representation of words
    - lda2vec: jointly learns word, document, topic vectors

Basic assumptions

- Semantics of our document are governed by hidden (latent) variables: **topics**
- Each **document** consists of a mixture of **topics**
- Each **topic** consists of a collection of **words**

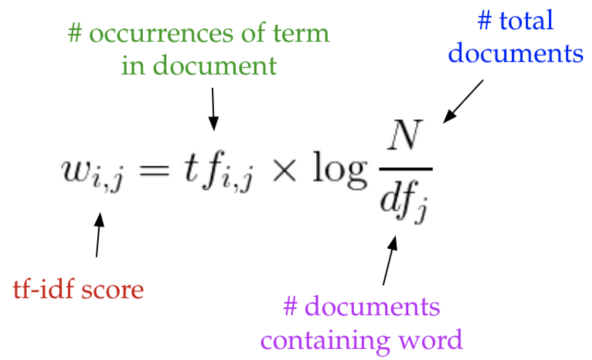


Latent Semantic Analysis (recap)

- Given m document and n words, build an *mxn document-term* matrix *A*

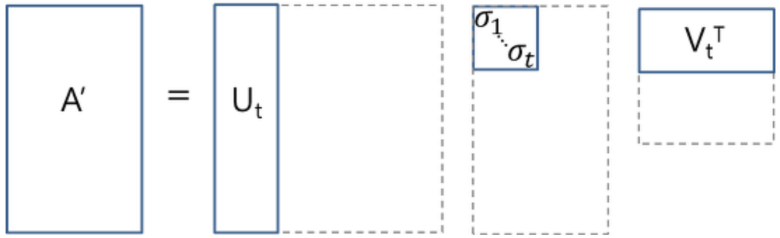
	a	an	apple	ate	banana	eat	i	today	will	yesterday
Doc 1		0.0811	0.0811	0.0811			0			0.0811
Doc 2		0.0676	0.0676			0.1831	0	0.1831	0.1831	
Doc 3	0.2197			0.0811	0.2197		0			0.0811

- Values $w_{i,j}$: weight of term *j* in document *i* (e.g. *tf.idf*):



- Matrix A is very sparse, very noisy, high-dimensional (and redundant)
- To find t topics, compute a *low rank approximation* of rank t
- *Truncated SVD*: compute SVD, keep t largest singular values, first t columns

$$A \approx U_t S_t V_t^T$$



- Document-topic matrix U_t: represents documents as vector of t topics
- Term-topic matrix V_t: represents terms as vector of t topics

- Benefits:
  - Embeds terms and documents in t-dimensional vector space
  - Use e.g. cosine similarity to compute:
    - similarity between documents
    - similarity between terms
    - similarity between query terms and (parts of) documents
- Drawbacks:
  - Embedding is not interpretable (what do topics mean?)
  - Needs really large set of documents and dictionary

Simple implementation (sklearn)

```
# Build document-term matrix
    vectorizer = TfidfVectorizer(stop_words='english',
                                 use_idf=True, smooth_idf=True)
    # Truncated SVD
    svd_model = TruncatedSVD(n_components=100, # Has to be tuned
                             algorithm='randomized')
    # Compute
    vectorized = vectorizer.fit_transform(documents)
    docs = svd_model.fit_transform(vectorized)    # Computes U_t * S_t
    terms = svd_model.fit_transform(vectorized.T) # Computes V_t * S_t
```

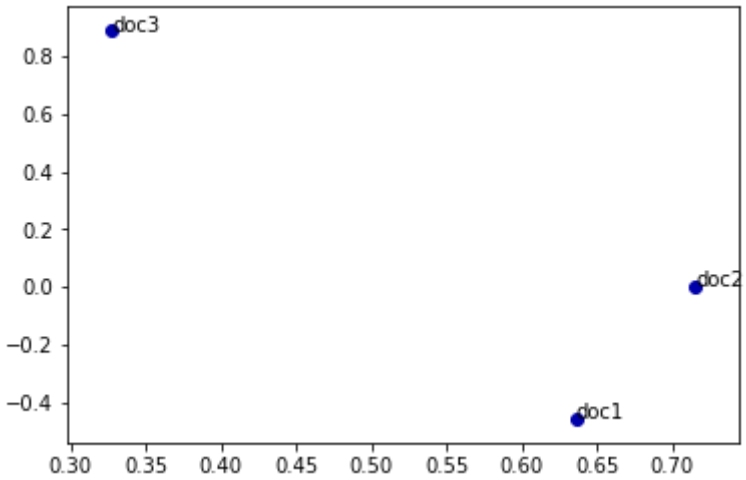
Example (BBC News 11/03/2018)

doc1: America's new commercial astronaut capsule has completed its demonstration flight with a successful splashdown in the Atlantic Ocean. The SpaceX Dragon vehicle left the International Space Station after being docked there for the past week, and re-entered Earth's atmosphere. It had a heat-shield to protect it from the high temperatures of re-entry. Four parachutes brought it into soft contact with water about 450km from Cape Canaveral, Florida. The mission - which had no humans aboard, only a dummy covered in sensors - went according to plan. The Dragon has set the stage for the US space agency Nasa to approve the vehicle for crewed flights."

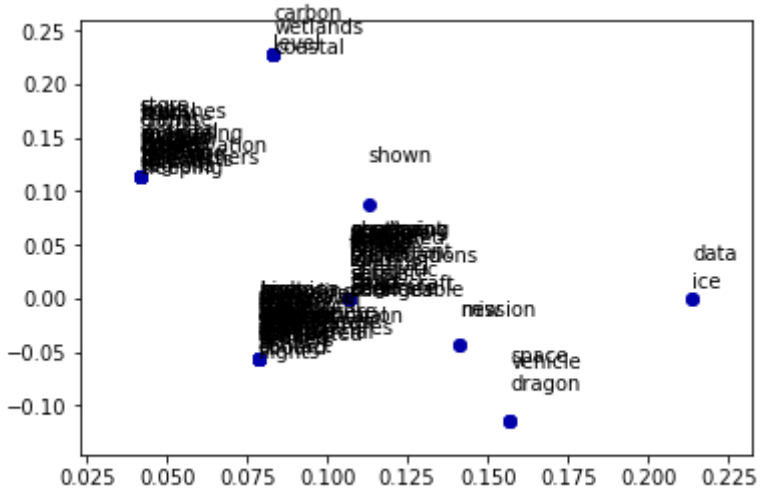
doc2: "Europe's quarter-century-long satellite data record of ice sheet changes in the Antarctic is secure into the future. It is possible because a new spacecraft tasked with observing what is going on in the polar south is finally producing very serviceable data. When the EU's Sentinel-3 mission was first launched in 2016, it struggled even to sense the continent's edge. That problem's been fixed and its observations of the ice have now been shown to be consistent and accurate."

doc3: "Muddy, coastal marshes are sleeping giants that could fight climate change, scientists say. A global study has shown that these regions could be awoken by sea level rise. Sea level is directly linked to the amount of carbon these wetlands store in their soil, the team reports in the journal Nature.

Vectorized documents (in 2 dimensions/topics for plotting):



Vectorized terms:


carbon
wetlands
coastal
shown
data
ice
dragon
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225

Probabilistic Latent Semantic Analysis (pLSA)

LSA:

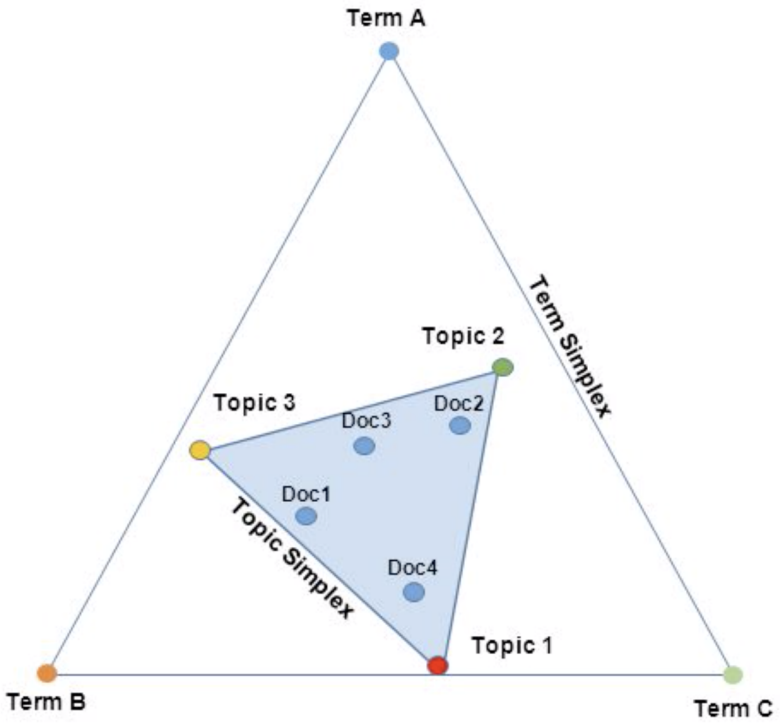
- Each **document** consists of a collection of **topics**
- Each **topic** consists of a collection of **words**

pLSA:

- Each **document** is a (multinomial) *probability distribution* over **topics**
  - E.g. 20% about topic 1, 50% topic 2, 30% topic 4
- Each **topic** is a (multinomial) *probability distribution* over **words**
  - E.g. 40% about word 1, 50% topic B, 30% topic C

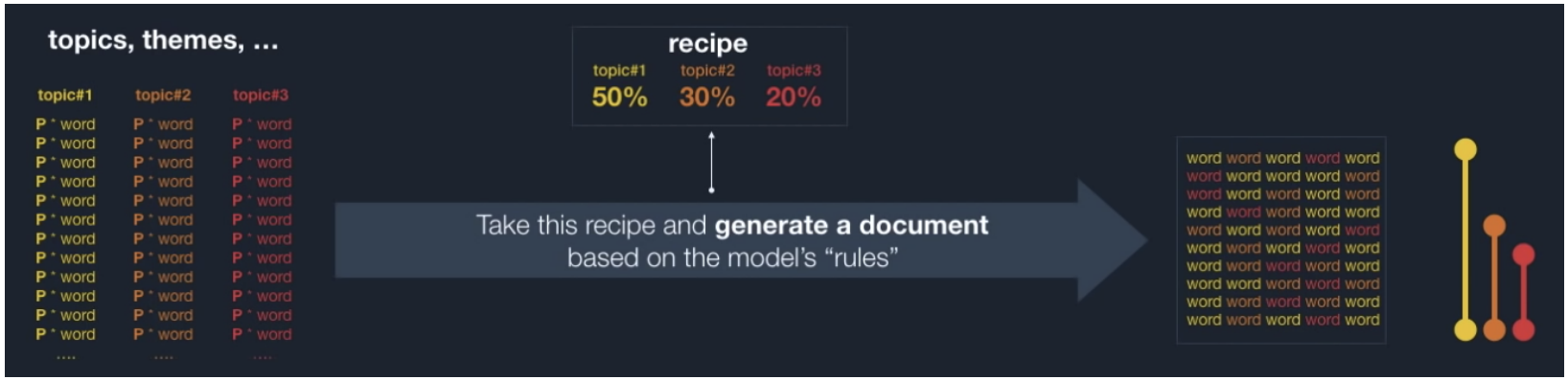
Can be represented on a simplex with barycentric coordinates:

- Doc 1: 30% about topic 1, 10% topic 2, 60% topic 3
- Topic 1: 5% about term A, 40% term B, 55% term C

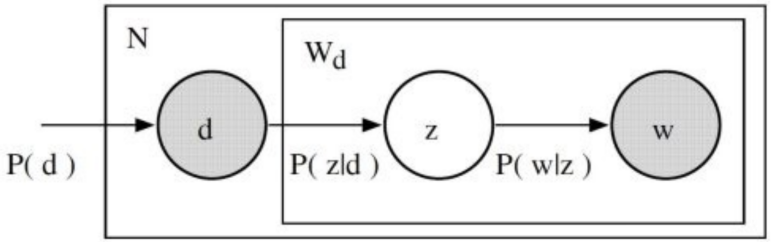


pLSA: Generative model

- Assumes that documents were generated by a (topic) model:
  - Choose a number of words in the document
  - Choose a topic distribution (*mixture*) over a fixed set of topics
  - Generate documents by:
    - Picking a topic z according to document's topic distribution $P(z|d)$
    - Picking a word w according to topics's word distribution $P(w|z)$
- Ignorant of syntax/grammar: just vocabulary words



Formally:

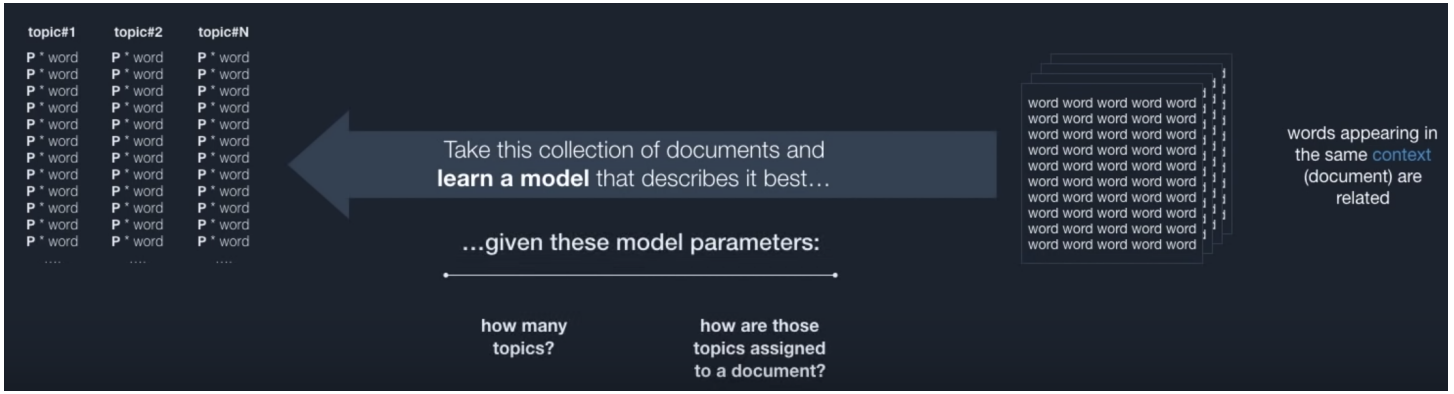


- Joint probability of seeing d and w together:

$$P(D, W) = P(D) \sum_{Z} P(Z|D)P(W|Z)$$

pLSA: Inference (expectation maximization)

- Reverse process: Given a corpus of documents,
  - learn the topic representation of *t* topics in each document and word distribution in each topic
  - Backtrack from documents to identify topics that generated the corpus
- Randomly assign each word to one of *t* topics
- For each document d, assume all *other* topic distribution are correct, and calculate:
  - p(z|d): proportion of words in *d* assigned to topic *z*
  - p(w|z): proportion of assignments to topic *z* over all documents with word *w*
- Assign *w* to topic *z* with probability p(z|d) * p(w|z), repeat



Equivalent representations:

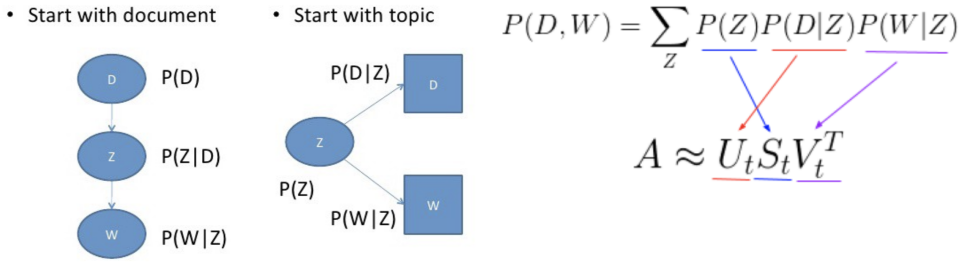
- Start with document with P(D), then generate topic with $P(z|d)$, then generate word with $P(w|z)$

$$P(D, W) = P(D) \sum_Z P(Z|D)P(W|Z)$$

- Start with topic with P(Z), then independently generate document with P(d|z) and word with $P(w|z)$

$$P(D, W) = \sum_Z P(Z)P(D|Z)P(W|Z)$$

- Now we see the parallel with LSA:



pLSA Benefits

- Neat way to explore / understand corpus
  - e-Discovery: which emails/tweets are relevant?
  - Social media: what are people saying about X?
  - Scientific data: what are main topics in literature
- Many applications:
  - Word sense disambiguation
  - Discourse segmentation
  - Machine translation
  - Object localization in images
  - Harmonic analysis of music
  - ...
- Handles polysemy (many meanings for same word)
- Inference is relatively dimple

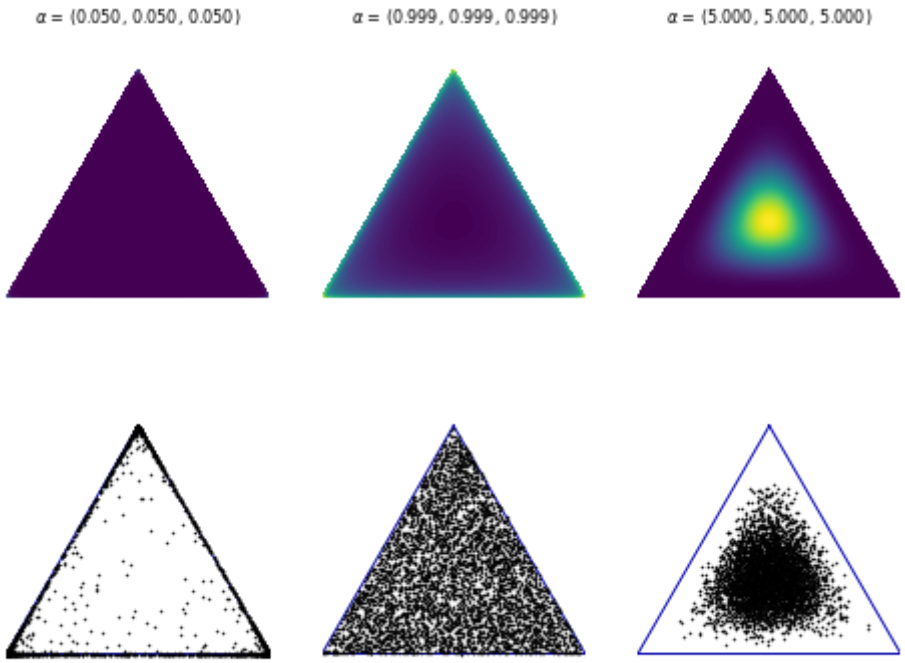
pLSA Drawbacks:

- We have no model parameters to model $P(D)$, we don't know how to assign probabilities to documents
- The number of parameters grows linearly with the number of documents: prone to overfitting

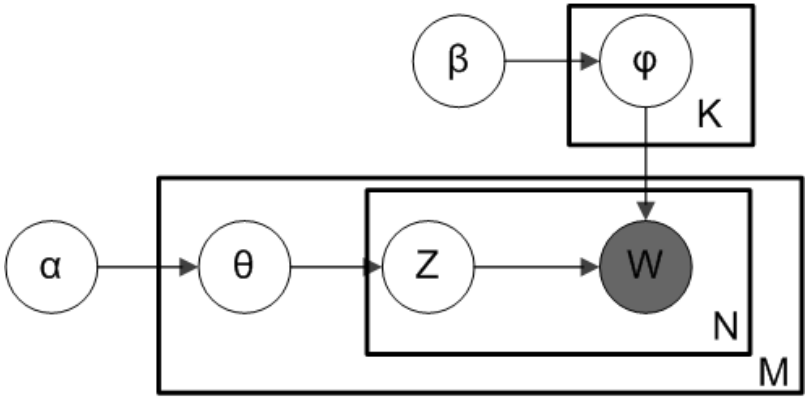
Latent Dirichlet Allocation (LDA)

- Bayesian version of pLSA, solves remaining drawbacks
- Uses *Dirichlet prior* for $P(z|d)$
- Dirichlet distribution is a distribution over our earlier multinomial distributions
- Hyperparameter α (can be different per topic):
  - Low α: document belongs to only few topics.
  - $\alpha = 1$: everything equally likely
  - High α: document belongs to many (or all) topics.

Sampling from this distribution, assuming 3 topics (full code in notebook):

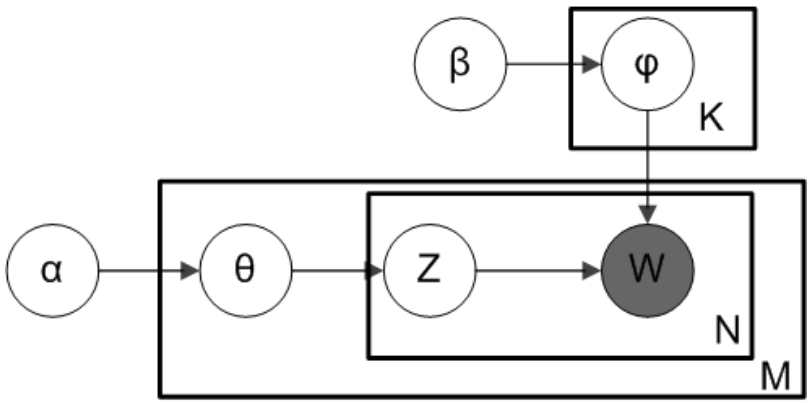


- We also define a Dirichlet distribution over $P(w|z)$
- Equivalent hyperparameter β:
  - Low β: every topic has only few words.
  - High β: every topic has many words.
- Formally, with topic distribution θ and word distribution φ:
  - for K topics, N words, M documents



LDA as a generative model

- Sample distribution θ_i from Dir(α)
  - θ_i, k: probability that document i has topic k)
- Sample distribution φ_k from Dir(β)
  - φ_k, w: probability that topic k has word w
- Sample topic z from θ_i
- Sample word w from φ_k



LDA inference

- Inference is similar to what we saw for pLSA
- For every new document, we can sample from the Dirichlet distribution
  - Hence, generalizes better to new documents
- We still need to choose the number of topics and the α,β hyperparameters

Examples

Many implementations, most useful:

- gensim
- sklearn.decomposition.LatentDirichletAllocation

gensim needs to be installed. We also add wordcloud to visualize the topics. nltk is usedul for preprocessing (e.g. stemming)

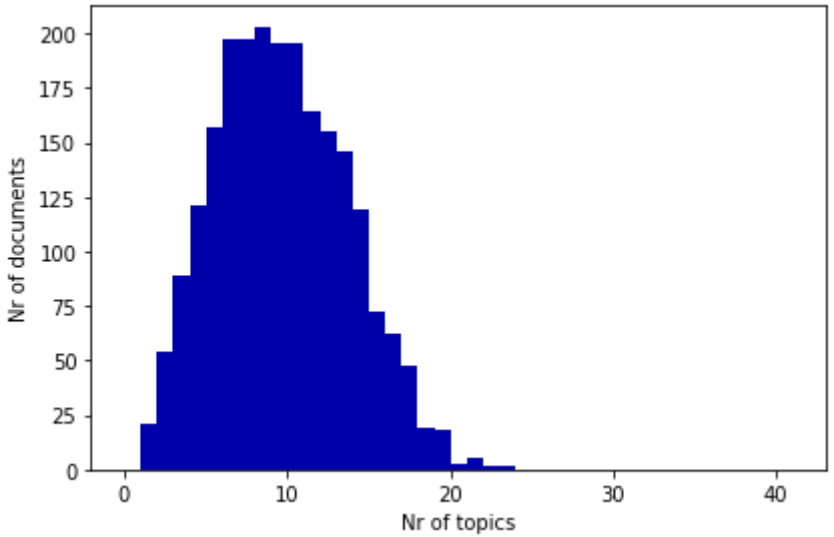
```
pip install gensim
pip install wordcloud
pip install nltk
```

News reports

- Data from the Associated Press (AP), already in the data folder
- Load a representation via `corpora.BleiCorpus`
- Use `models.ldamodel.LdaModel` to run LDA

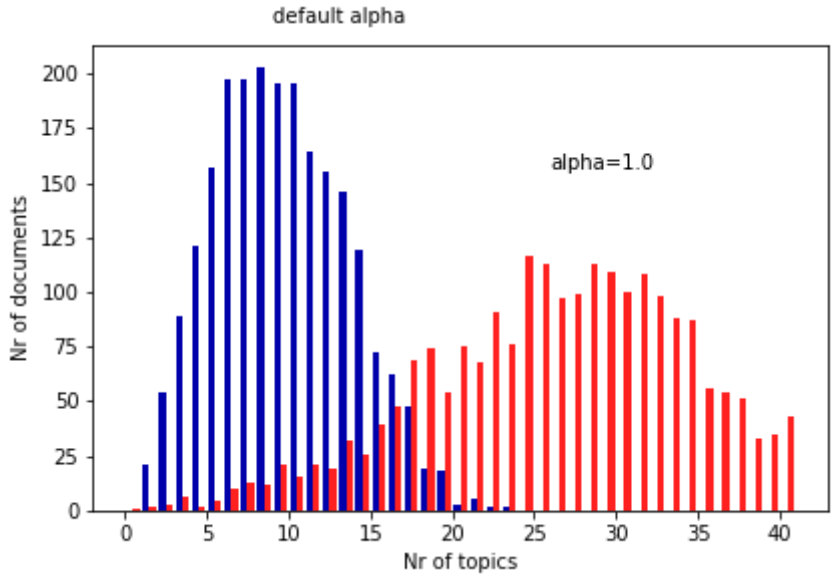
```
model = models.ldamodel.LdaModel(
    corpus, num_topics=100,
    id2word=corpus.id2word, alpha=None) # default alpha
```

- How many topics does every document have?



We can do the same after changing the α value:

```
[WARNING] [12:42:33:gensim.models.ldamodel] too few updates, training migh
t not converge; consider increasing the number of passes or iterations to
improve accuracy
```

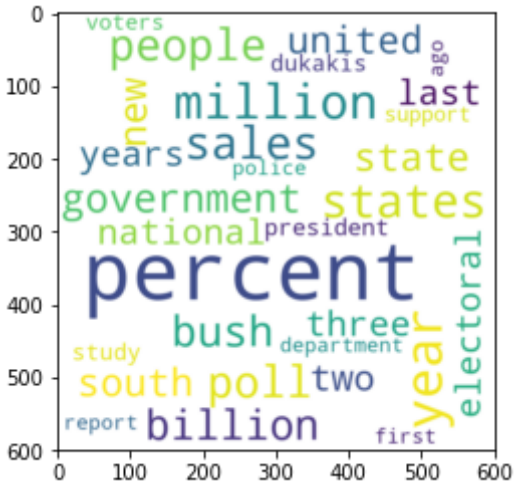


What is the distribution of topics in the first document?

```
[(7, 0.048612498), (8, 0.015723428), (31, 0.35472104), (34, 0.13508442),
(38, 0.13842407), (55, 0.039698716), (60, 0.013031449), (63, 0.041657068),
(76, 0.011997577), (84, 0.054972194), (86, 0.0135803055), (88, 0.10172871
5)]
```

A better way to explore the data is to identify the most discussed topic, i.e., the one with the highest total weight.

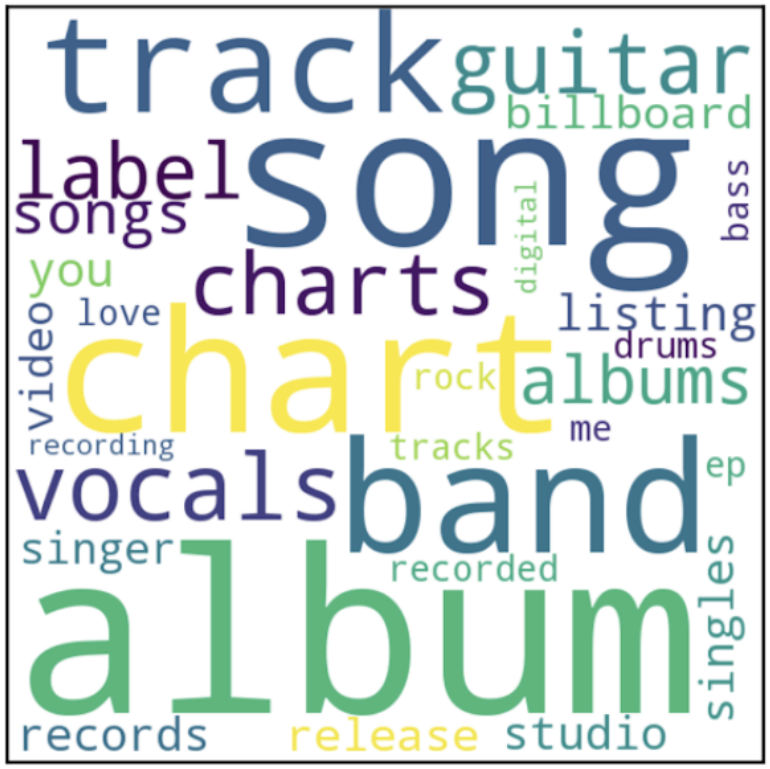
- Get the top 64 words for this topic.
- Use `max_words` to set the number of words we want (default is 10).
- Visualize the results in a word cloud



Modeling Wikipedia

- We can do the same on a large set of articles from Wikipedia.
- Execute the `download_wp` script in the data folder.
  - Downloads and preprocessed the data.
  - Will take a few hours and over 14GB of disk space.

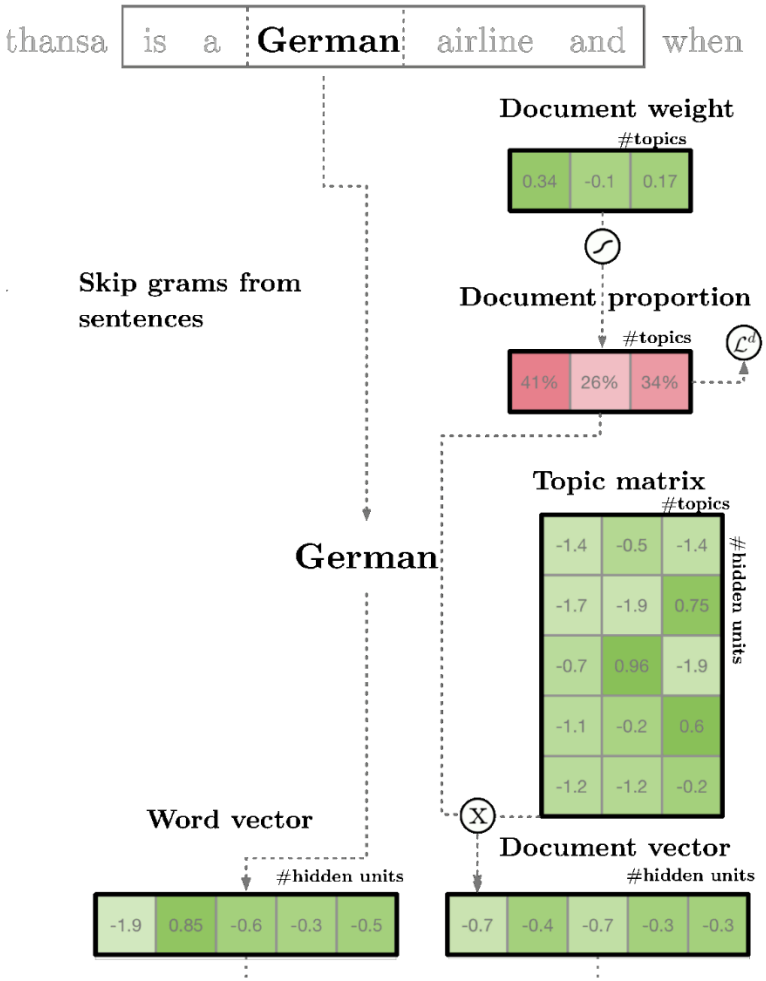
Retrieve the most heavily used topic and plot it as a word cloud:

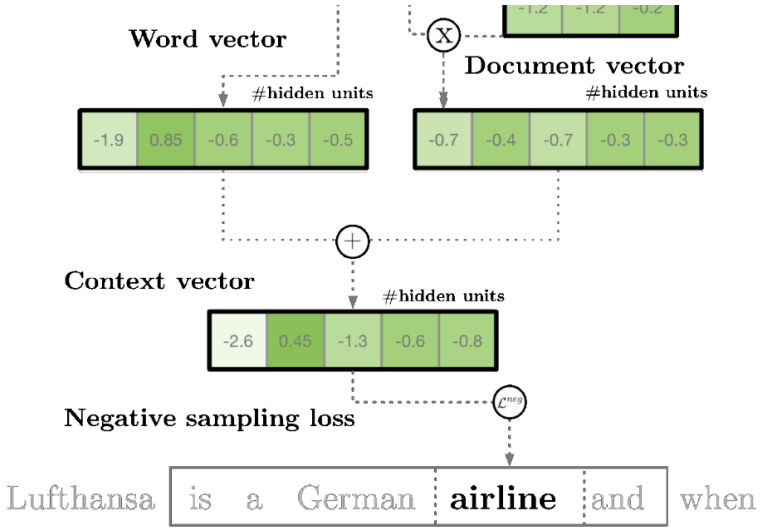


LDA in Deep Learning: lda2vec

- lda2vec is an extension of word2vec and LDA that jointly learns word, document, and topic vectors.
- builds on top of the skip-gram model of word2vec to generate word vectors.
  - Neural net that learns a word embedding by trying to use the input word to predict surrounding context words
  - See http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ (http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)
- instead of using the word vector directly to predict context words, we leverage a context vector to make the predictions.
- this context vector is created as the sum of two other vectors: the word vector and the document vector.

- The document vector is a weighted combination of two other components:
  - the document weight vector, representing the “weights” of each topic in the document
  - the topic matrix, representing each topic and its corresponding vector embedding
- Together, the document vector and the word vector generate “context” vectors for each word in the document.
- lda2vec not only learns word embeddings (and context vector embeddings) for words,
- it simultaneously learns topic representations and document representations


thansa is a German airline and when
Document weight
#topics
0.34 -0.1 0.17
Skip grams from sentences
Document proportion
#topics
41% 26% 34%
$\mathcal{L}^d$
Topic matrix
#topics
-1.4 -0.5 -1.4
-1.7 -1.9 0.75
-0.7 0.96 -1.9
-1.1 -0.2 0.6
-1.2 -1.2 -0.2
#hidden units
German
Word vector
#hidden units
-1.9 0.85 -0.6 -0.3 -0.5
Document vector
#hidden units
-0.7 -0.4 -0.7 -0.3 -0.3



Further reading on lda2vec

- blog post (https://multithreaded.stitchfix.com/blog/2016/05/27/lda2vec/#topic
- Code (https://github.com/cemoody/lda2vec)
- Example (http://nbviewer.jupyter.org/github/cemoody/lda2vec/blob/master/examples/