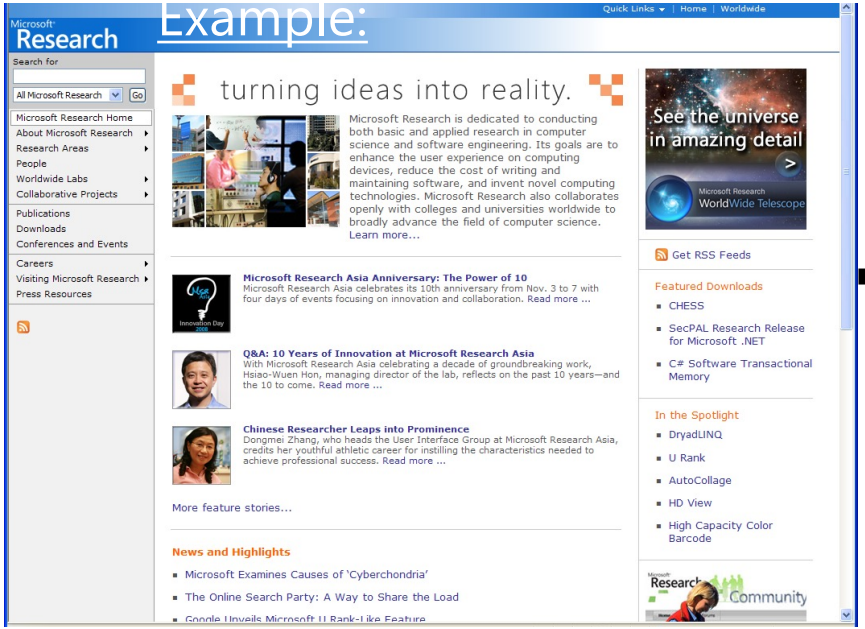
Methods for assigning topics to content

- Human tagging
  - Perhaps with some automated curation…
- Supervised learning
  - Train a classifier with existing tagged data (e.g. OpenDirectory Web pages: 210 topics)
  - Tag new documents with the most likely classes as predicted by the classifier.
- Unsupervised learning
  - Gather a corpus of documents (unlabeled).
  - Apply a topic modeling algorithm.
  - Try to interpret the topics that result!
- Semi-supervised topic modeling (e.g. Labeled LDA)

Supervised method: classifying Web pages into topics

Analysis of homepage @ research.microsoft.com

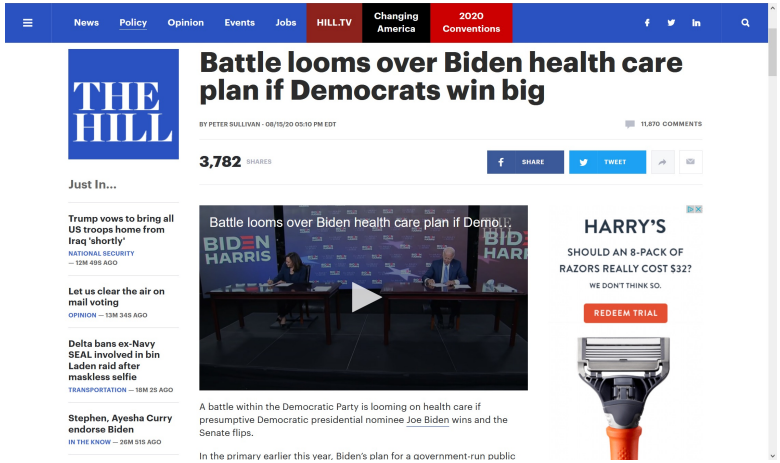


Classes
(from OpenDirectory)

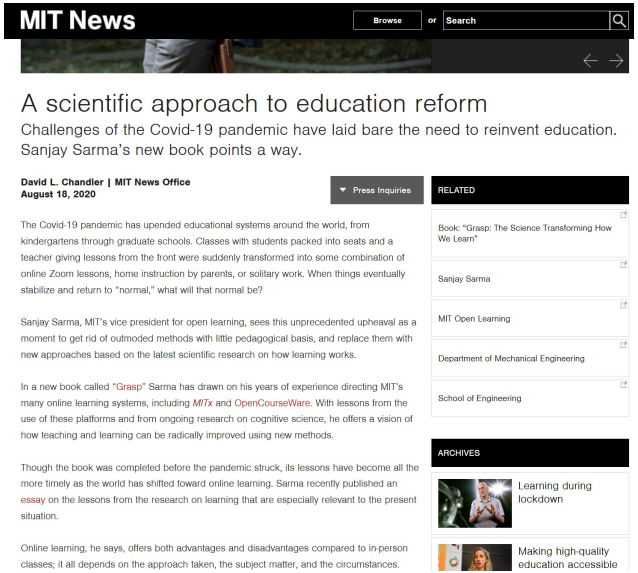
- 0.23 computers
- 0.10 science
- 0.08 society/issues
- 0.07 reference
- 0.07 reference/education

A single document can be associated with multiple topics

Politics or health?



Science or education?



Application: Discovering dominant themes in a collection.
Sample scientific topics discovered with Latent Dirichlet Allocation

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
computer	chemistry	cortex	orbit	infection
methods	synthesis	stimulus	dust	immune
number	oxidation	fig	jupiter	aids
two	reaction	vision	line	infected
principle	product	neuron	system	viral
design	organic	recordings	solar	cells
access	conditions	visual	gas	vaccine
processing	cluster	stimuli	atmospheric	antibodies
advantage	molecule	recorded	mars	hiv
important	studies	motor	field	parasite

FIGURE 1. Five topics from a 50-topic LDA model fit to *Science* from 1980–2002.

Terms are usually ordered by their probability given the topic.

Source: Blei & Lafferty (2009) “Topic Modeling”, http://www.cs.columbia.edu/~blei/papers/BleiLafferty2009.pdf

Application: Social Media

Topic modelling applied to 4,170,382 tweets from 1,200 prominent Twitter accounts, posted over 12 months. Topics can be identified based on either individual tweets, or at the user profile level.

Topic 1

Rank	Term
1	space
2	#yearinspace
3	pluto
4	earth
5	nasa
6	mars
7	mission
8	launch
9	#journeytomars
10	science

Topic 2

Rank	Term
1	#health
2	cancer
3	study
4	risk
5	patients
6	care
7	diabetes
8	#zika
9	drug
10	disease

Topic 3

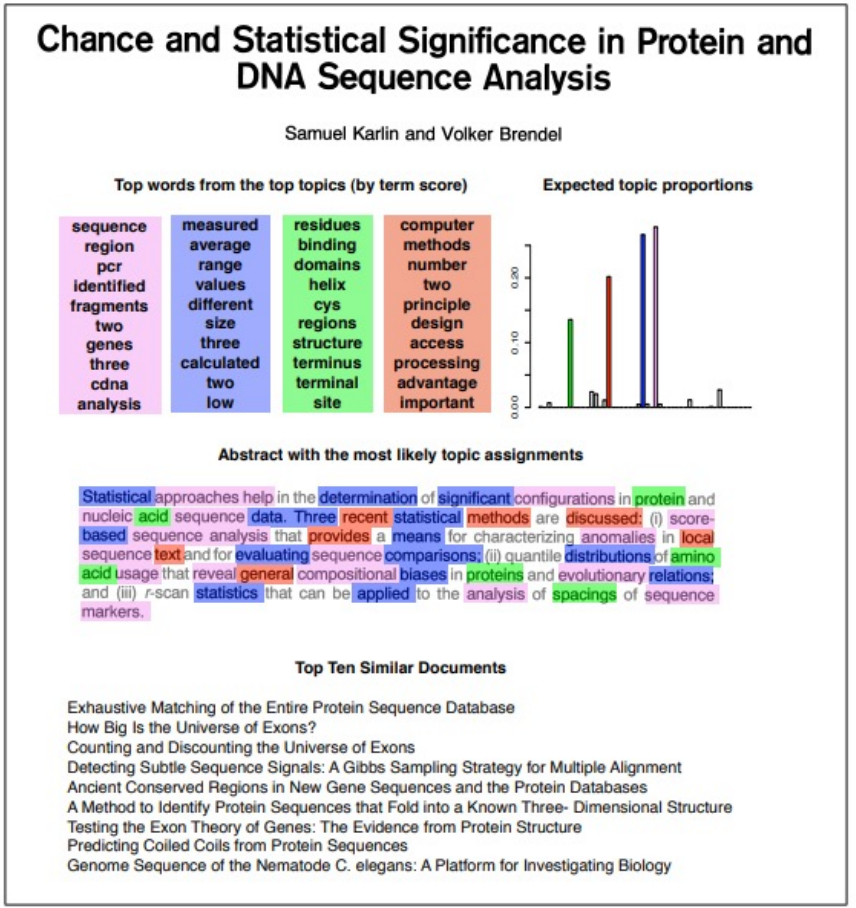
Rank	Term
1	apple
2	iphone
3	#ios
4	ipad
5	mac
6	app
7	watch
8	apps
9	os
10	tv

Source: http://derekgreene.com/slides/topic-modelling-with-scikitlearn.pdf

What is a topic model?

- An unsupervised ML technique for discovering the abstract themes that occur in a set of documents.
- A ‘document’ could be of many types:
  - A web page
  - A social media post
  - News or lecture video
  - DNA transcription
  - LEGO color themes

1. Topic proportions
2. Top scoring words from the most prevalent topics
3. The assignment of words to topics in the abstract of the article
4. Top ten most similar articles



Chance and Statistical Significance in Protein and DNA Sequence Analysis

Samuel Karlin and Volker Brendel

Top words from the top topics (by term score)

sequence	measured	residues	computer
region	average	binding	methods
pcr	range	domains	number
identified	values	helix	two
fragments	different	cys	principle
two	size	regions	design
genes	three	structure	access
three	calculated	terminus	processing
cdna	two	terminal	advantage
analysis	low	site	important

Expected topic proportions

Abstract with the most likely topic assignments

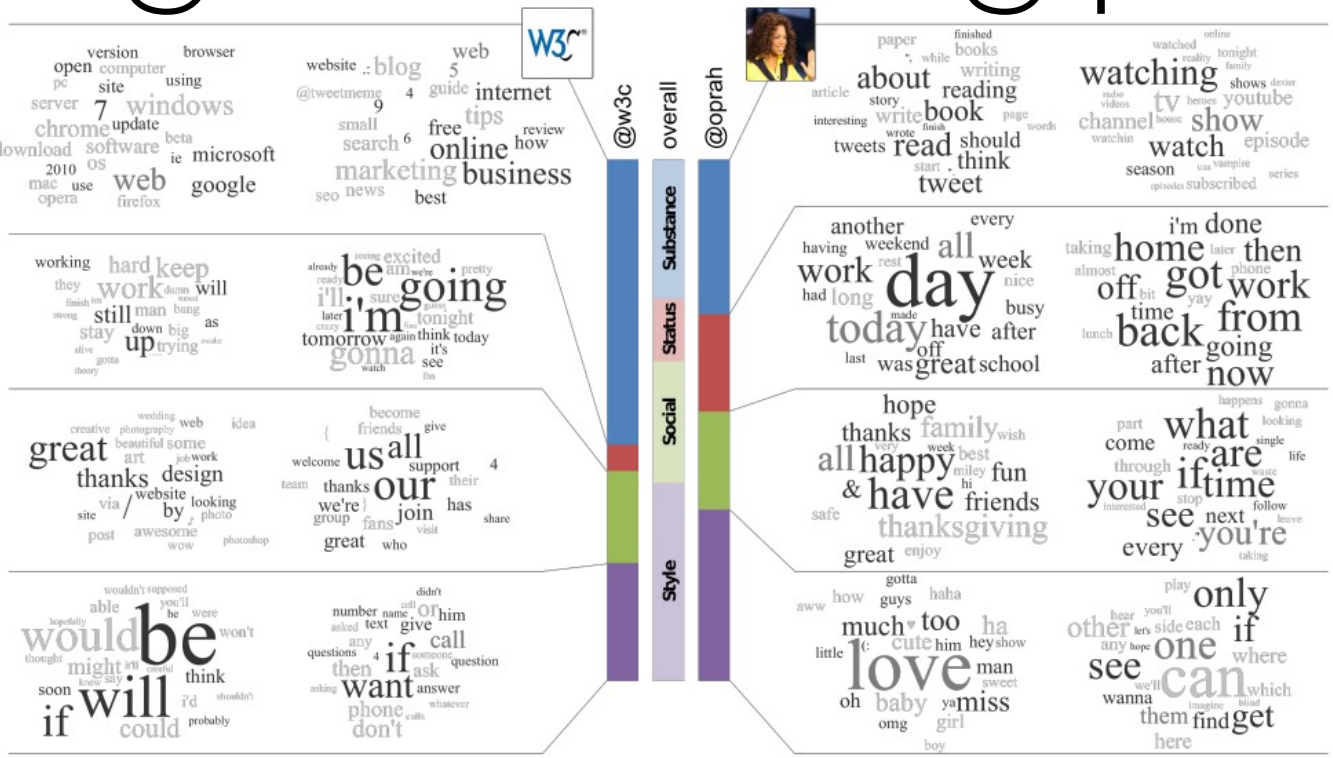
Statistical approaches help in the determination of significant configurations in protein and nucleic acid sequence data. Three recent statistical methods are discussed: (i) score-based sequence analysis that provides a means for characterizing anomalies in local sequence text and for evaluating sequence comparisons; (ii) quantile distributions of amino acid usage that reveal general compositional biases in proteins and evolutionary relations; and (iii) *r*-scan statistics that can be applied to the analysis of spacings of sequence markers.

Top Ten Similar Documents

Exhaustive Matching of the Entire Protein Sequence Database
How Big Is the Universe of Exons?
Counting and Discounting the Universe of Exons
Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment
Ancient Conserved Regions in New Gene Sequences and the Protein Databases
A Method to Identify Protein Sequences that Fold into a Known Three- Dimensional Structure
Testing the Exon Theory of Genes: The Evidence from Protein Structure
Predicting Coiled Coils from Protein Sequences
Genome Sequence of the Nematode C. elegans: A Platform for Investigating Biology

Source: Blei & Lafferty (2009) “Topic Modeling”, http://www.cs.columbia.edu/~blei/papers/BleiLafferty2009.pdf

Topic modeling of tweets (Labeled LDA)



Source: Ramage, Dumais, Liebling (ICWSM 2010). Characterizing Microblogs with Topic Models

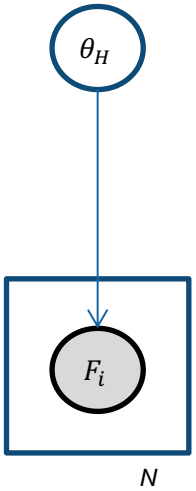
Two general approaches to topic modeling are most popular

- Probabilistic (e.g. Generative models)
  - Each topic is modeled as a distribution over words
  - Each document is modeled as a mixture of a few topics
  - Example: **Latent Dirichlet Allocation** (LDA)
- Matrix factorization
  - Decompose a document-term matrix into a small number of matrix factors (usually 2)
  - The resulting columns in a factor can be interpreted as a topic model.
  - Example: **Non-negative matrix factorization** (NMF)

Unsupervised topic models: generative processes

- Imagine we already have a set *T* of *K* topic models, each of which is a probability distribution over words, denoted β_k. (assume fixed vocabulary of *V* possible words)
  - We want to model documents as being mixtures of different topics.
  - How could we view a document as being generated using *T*?
- Suppose we want to generate document *d* that should be *n* words long.
- Here's one way we could write a program to fill the *n* word slots:
- For each document *d:*
  - Randomly pick a vector θ_d of topic proportions over T. (computers 0.23, science 0.10, society 0.05, …)
  - For *i = 1 to n* (each of the *n* word positions in *d*), fill the word slot with these two steps:
    - Randomly pick a topic $Z_{d,i} \in \{1 \ldots K\}$ based on the document's topic proportion vector θ_d
    - Fill the word slot by drawing a word randomly from the corresponding topic model for topic $Z_{d,i}$

Generative processes can be visualized graphically using ‘plate notation’



Graphical model of simple coin-flipping:
A coin with probability θ_H of Heads is flipped *N* times.

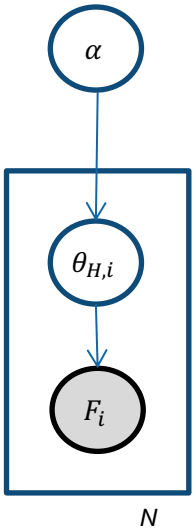
Variables:
θ_H : parameter p(heads)
F_i : outcome of i-th flip

θ_H is determined once at the start and then used for all N flips.

Each F_i depends on θ_H.

- Each circular node is a variable.
- Variable A points to variable B, if B depends on A.
- A rectangle (or ‘plate’) shows variables that repeat together.
- Each plate is labelled with the number of repetitions.
- Shaded variables represent observable quantities (in this case, words).
- Unshaded variables represent latent variables/parameters.

Generative processes can be visualized graphically using ‘plate notation’



Variables:
$\theta_{H,i}$: per-flip p(heads)
F_i : outcome of i-th flip
α: parameter controlling distribution that we select $\theta_{H,i}$ from.

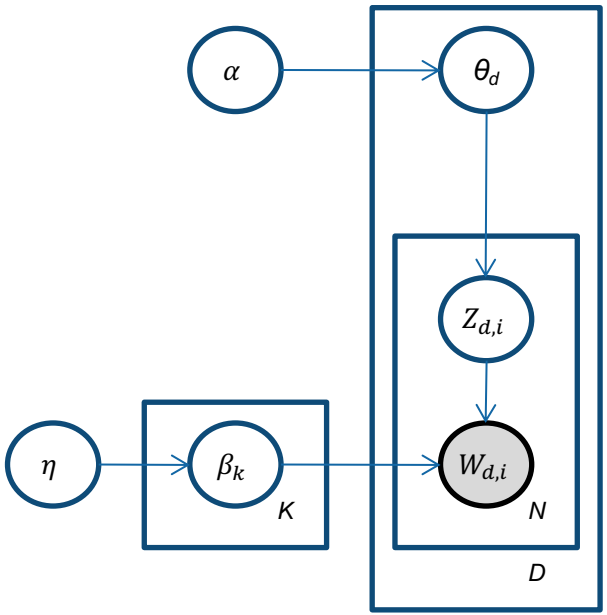
A new $\theta_{H,i}$ is determined for each flip.

Each F_i still depends on $\theta_{H,i}$.

Graphical model of bizarre coin-flipping:
A coin with probability $\theta_{H,i}$ of Heads that changes on every flip.

- Each circular node is a variable.
- Variable A points to variable B, if B depends on A.
- A rectangle (or ‘plate’) shows variables that repeat together.
- Each plate is labelled with the number of repetitions.
- Shaded variables represent observable quantities (in this case, words).
- Unshaded variables represent latent variables/parameters.

Here's a graphical model for a generative topic model



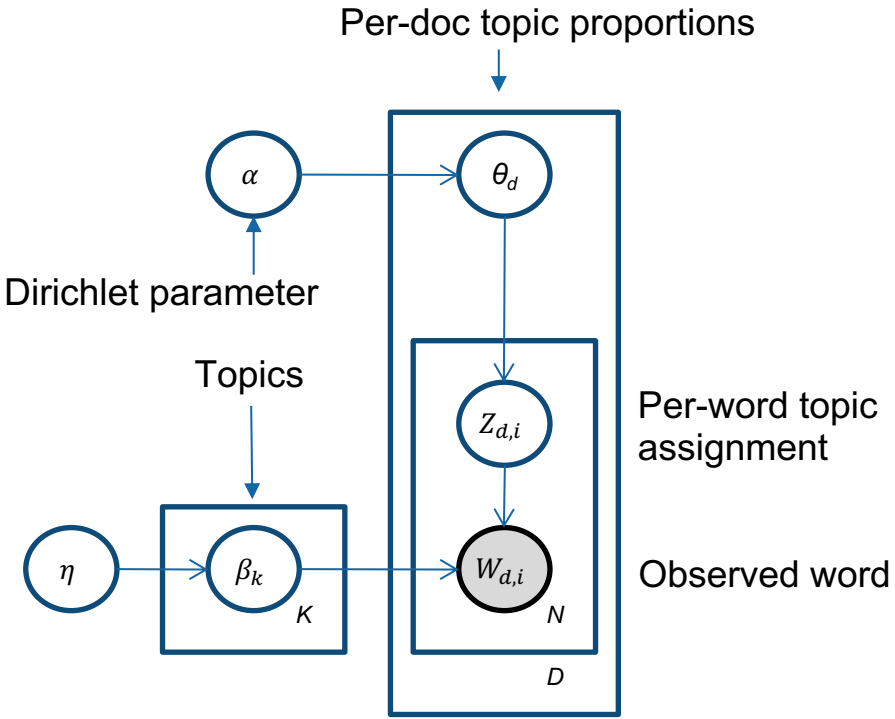
- Each circular node is a variable.
- Variable A points to variable B, if B depends on A.

- A rectangle (or 'plate') shows variables that repeat together.
- Each plate is labelled with the number of repetitions.

- Shaded variables represent observable quantities (in this case, words).

- Unshaded variables represent latent variables/parameters.

Latent Dirichlet Allocation [Blei, Ng, Jordan. 2001]



- Generative model
- A topic (equiv. topic model) is a distribution over words.
- Each word modeled as a sample from mixture model (of topics)
- Each word is generated from a single topic
- Different words in a doc may be generated from different topics.
- Thus a document is not limited to exhibiting one topic only.

Topics are defined by the K parameter vectors

$$\beta_1, \beta_2, \ldots, \beta_K$$

Per-document topic proportions are defined by the D parameter vectors

$$\theta_1, \theta_2, \ldots, \theta_D$$

Per-word topic assignments defined by

$Z_{d,i}$ for *d* = 1..D and *i* = 1.. N

Binomial Distribution

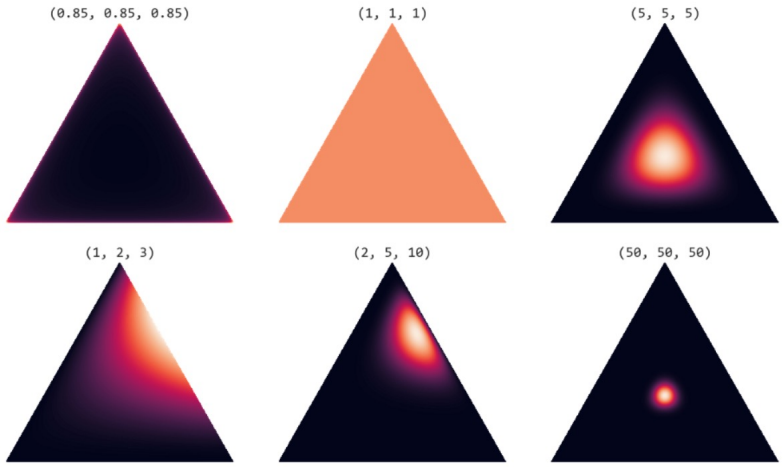
- n independent trials
- Each trial has 2 possible outcomes (binary).
- What is the probability of observing any particular combination of outcomes?
- Parameters: a number θ the probability of seeing the first type of outcome.
- Example: Flipping a coin c, with probability θ of being Heads.
- Denoted by $c \sim B(n, \theta)$

Multinomial Distribution
Generalizes the Binomial Distribution to *k* categories (from 2)

- n independent trials
- Each trial picks an object from one of k categories.
- What is the probability of observing any particular combination of objects from the various categories?
- Parameters: a vector θ of *k* probabilities (that sum to 1)
  - The i-th element $\theta[i]$ of θ, is the probability of selecting an object from the i-th category
- Example: Words *w* chosen from a fixed vocabulary
- Denoted by $w \sim Mult(n, \theta)$

Dirichlet Distribution

- Allows us to specify a probability distribution over multinomials.
- It's a distribution over the simplex, *k* positive numbers that sum to 1.
- Assumes components are *nearly* independent
- What is the probability of observing a given multinomial distribution over *k* categories?
- Parameters: vector α of *k* real numbers
- Denoted by $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha})$

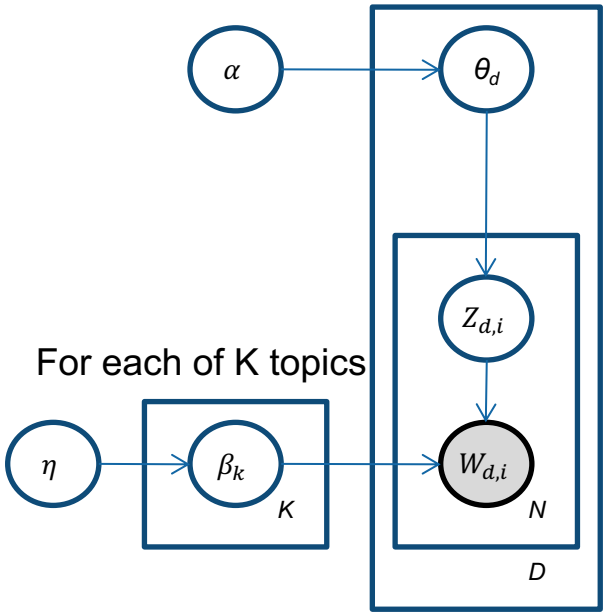


Dirichlet distribution on a 2-simplex (equilateral triangle) for different values of α.

Multinomial parameter vector (k elements)

Source: https://github.com/yusueliu/medium/blob/master/scripts/plot_dirichlet.py

Latent Dirichlet Allocation [Blei, Ng, Jordan. 2001]



- We said "randomly pick" but didn't say
using which types of distributions.
So let's get specific:

$$\theta_d \sim \text{Dirichlet}(\alpha)$$
$$\beta_k \sim \text{Dirichlet}(\eta)$$
$$z \sim \text{Multinomial}(\theta)$$
$$w \sim \text{Multinomial}(\beta_k)$$

- Topics and words can vary in generality

So where do the final topics come from? LDA estimates the model parameters that best 'explain' the observed words in the corpus, given the model.

- Given the observed corpus, in theory we want to infer the posterior distribution over the latent variables.
- The posterior is sort of a 'reversal' of the generative process: which latent topic structure most likely generated the corpus we observe?
- LDA estimates the posterior expectations of the latent variables:
  - The K topics $\beta_1, \beta_2, \ldots, \beta_K$
  - Per-document topic proportions $\theta_1, \theta_2, \ldots, \theta_D$
  - Topic assignment of each word $Z_{d,i}$ for *d* = 1..D and *i* = 1.. N

- You must supply the # of topics *K* as an input parameter.

So where do the topics come from? LDA estimates the model parameters that best 'explain' the observed words in the corpus, given the model.

- We could also learn the hyperparameters α, η to get information about the corpus:
  - α : Semantic diversity of docs
  - η : How similar topics are
  - θ : Prob. of each topic in each document
- Computing these are beyond the scope of this course…
  - Mean field variational inference (default), Expectation propagation, Gibbs sampling

Scikit-learn has an LDA implementation

- Inputs:
  - *X* : document-term frequency count (tf) matrix (not tf.idf)
  - The number of topics k (property: n_components) must be specified.
  - Lots of other tuning parameters but usually default settings are fine.
  - For smaller collections could use grid search to search for lowest perplexity model.
- Output:
  - `components_[i, j]` can be viewed as a pseudocount that represents the number of times word *j* was assigned to topic *i.*
  - With default LDA results may vary each time with the same input.

```
from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n_components = n_topics, random_state=0)
lda.fit(X)
topic_models = lda.components_
```

Typical usage steps for scikit-learn LDA

1.
```
documents_train = ['the cat and the dog and the duck were friends.',
                   'computers have power supplies.',
                   'plug in the monitor and turn on the computer.',
                   'you will find the plug on the right side of the screen.',
                   'my friend likes coffee and cats.',
                   'his dog gets along well with my friend.']
```

2.
```
tf_vectorizer = CountVectorizer(stop_words='english')
tf_documents = tf_vectorizer.fit_transform(documents_train)
tf_feature_names = tf_vectorizer.get_feature_names()
```

[Simple use of CountVectorizer for clarity]

3.
```
lda = LatentDirichletAllocation(n_components = n_topics, random_state=0)
lda.fit(tf_documents)
topic_models = lda.components_
```

4.
```
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        term_list = [feature_names[i]
                     for i in topic.argsort()[:-no_top_words - 1:-1]]
        print("topic %d:" % (topic_idx), term_list)


display_topics(lda, tfidf_feature_names, num_top_words)
```

```
topic 0: ['dog', 'friend', 'cat', 'friends', 'duck', 'cats', 'likes', 'coffee']
topic 1: ['plug', 'computer', 'monitor', 'turn', 'screen', 'right', 'power', 'computers']
```

Applying LDA in scikit-learn: Getting topic document weights

```
# Use transform on the original document-term matrix
# to get the document weights per topic
lda_output = lda.transform(tf_documents)
print("LDA transform output:\n", lda_output)
best_document_per_topic = np.argsort(lda_output, axis = 0)[::-1]
for topic_index in range(0, 2):
    best_index = best_document_per_topic[0, topic_index]
    print("Highest topic", topic_index, "weight is document",
          best_index, ":", documents_train[best_index])
```

Documents

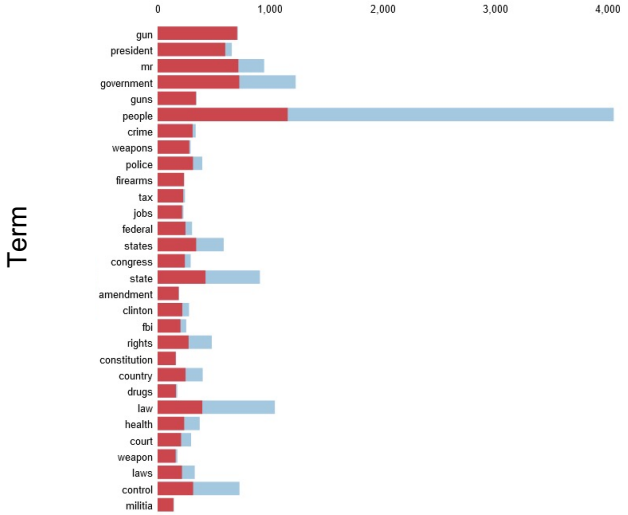
```
LDA transform output:
 [[0.89424792 0.10575208]
 [0.86647716 0.13352284]
 [0.1025136  0.8974864 ]
 [0.12801236 0.87198764]
 [0.89424791 0.10575209]
 [0.86960835 0.13039165]]
Highest topic 0 weight is document 0 : the cat and the dog and the duck were friends.
Highest topic 1 weight is document 2 : plug in the monitor and turn on the computer.
```

Topics

Applying LDA in scikit-learn: the 20 newsgroups dataset

Topics (K=10). Highlighting the 'white house' topic #4

```
topic 0: ['use', 'like', 'know', 'used', 'power', 'drive', 'new', 'don']
topic 1: ['just', 'like', 'don', 'good', 'think', 'time', 'car', 've']
topic 2: ['god', 'people', 'don', 'think', 'does', 'just', 'say', 'jesus']
topic 3: ['ax', 'max', 'g9v', 'b8f', 'a86', 'pl', '145', '1d9']
topic 4: ['people', 'government', 'mr', 'gun', 'president', 'don', 'think', 'know']
topic 5: ['edu', 'windows', 'use', 'file', 'com', 'available', 'software', 'program']
topic 6: ['space', 'file', 'university', 'program', 'nasa', '1993', 'israel', 'research']
topic 7: ['key', 'scsi', 'drive', 'chip', 'encryption', 'use', 'clipper', 'keys']
topic 8: ['game', 'team', 'year', 'db', 'armenian', 'said', 'people', 'armenians']
topic 9: ['00', '10', '25', '17', '11', '15', '16', '14']
```

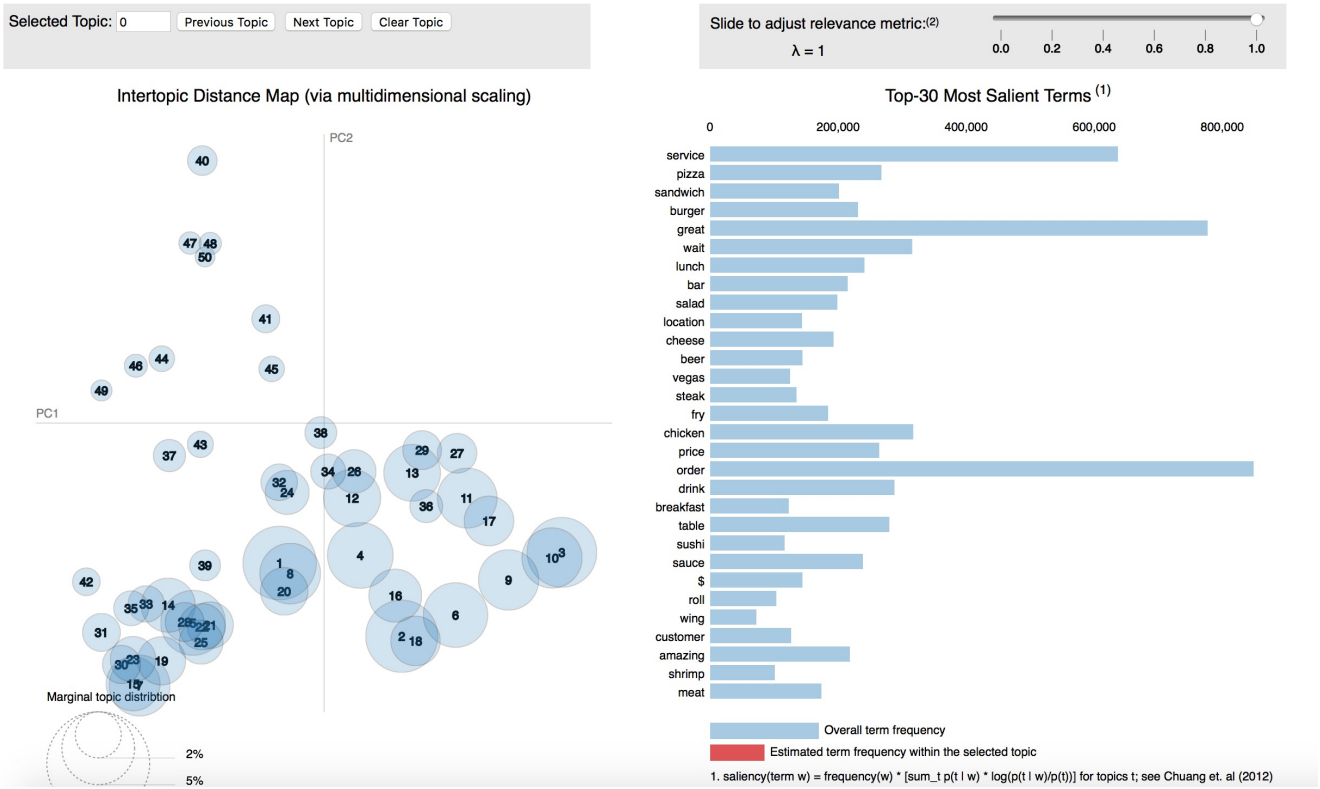


Term frequency (within topic /within collection)

Top documents per topic

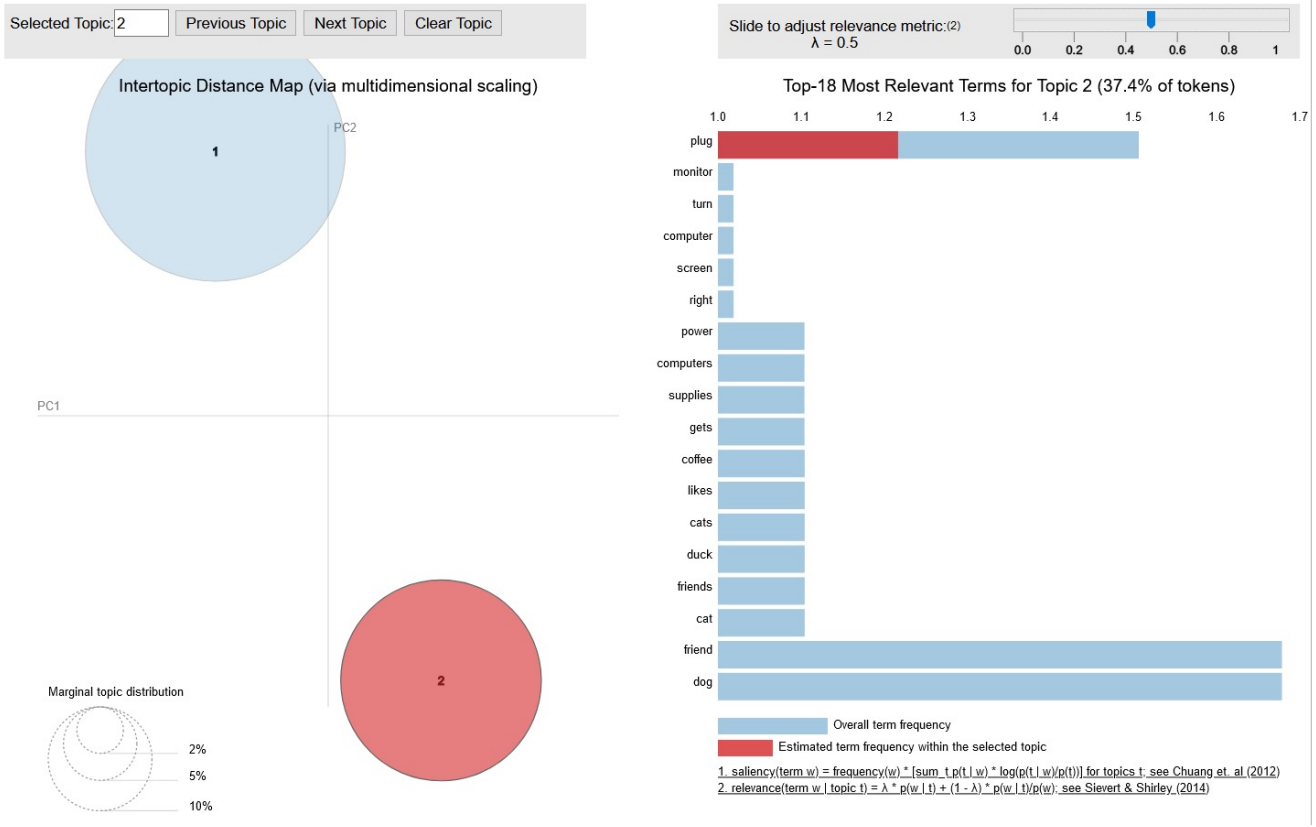
```
Highest topic 0 weight is document 2339 : House wiring and grounding
Highest topic 1 weight is document 1461 : Super slo-mo and frame advance
Highest topic 2 weight is document 5200 : An Introduction to Atheism
Highest topic 3 weight is document 4772 : ------------ TGZ Part 12 of 14
Highest topic 4 weight is document 6635 : White House Press Release
```

Visualizating topics with PyLDAVis

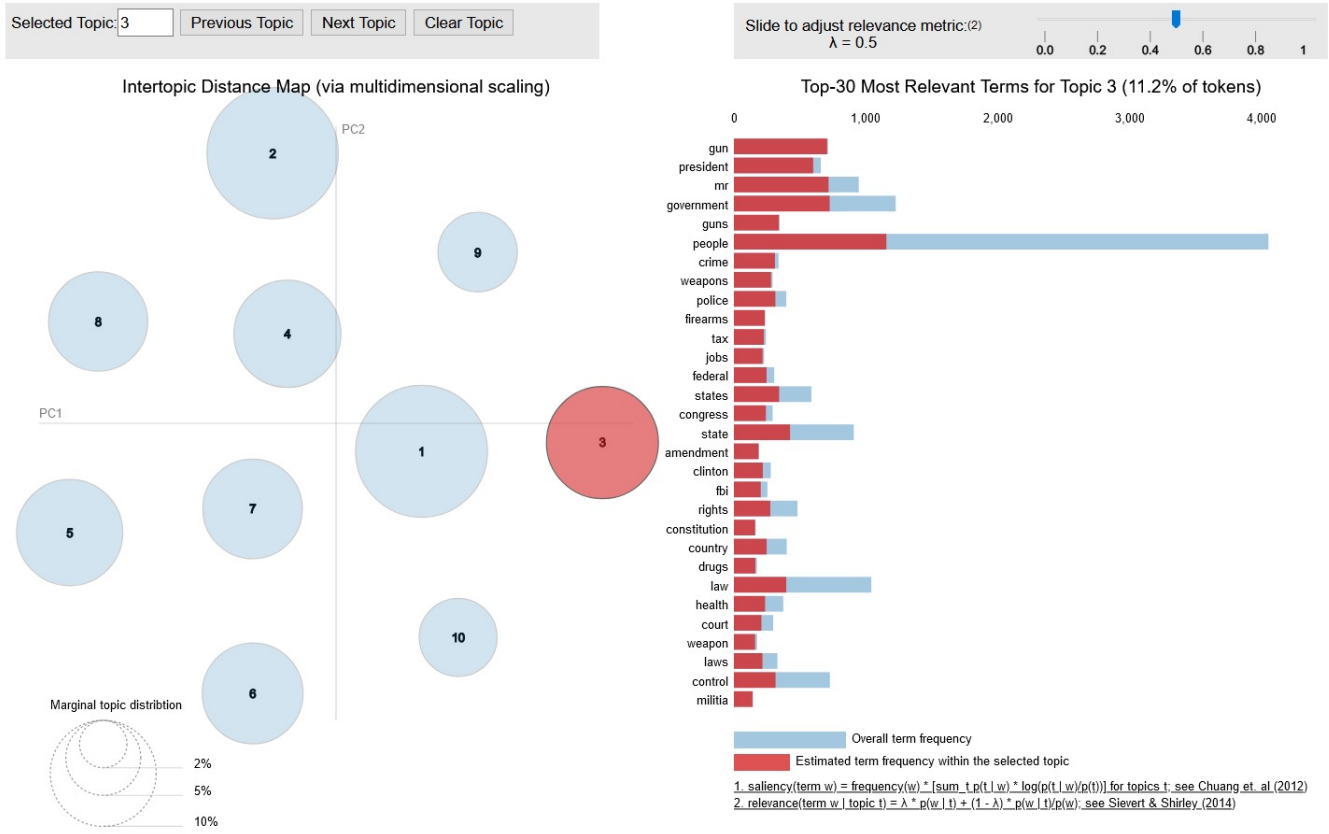


Source: https://github.com/bmabey/pyLDAvis

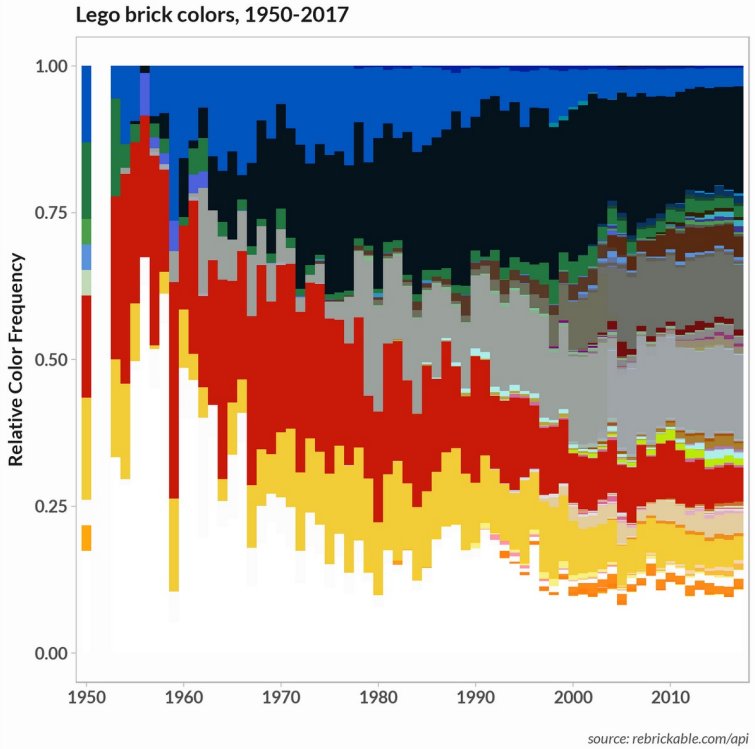
```
import pyLDAvis.sklearn

pyLDAvis.enable_notebook()
panel = pyLDAvis.sklearn.prepare(lda, tf_documents, tf_vectorizer, mds='tsne')
panel
```

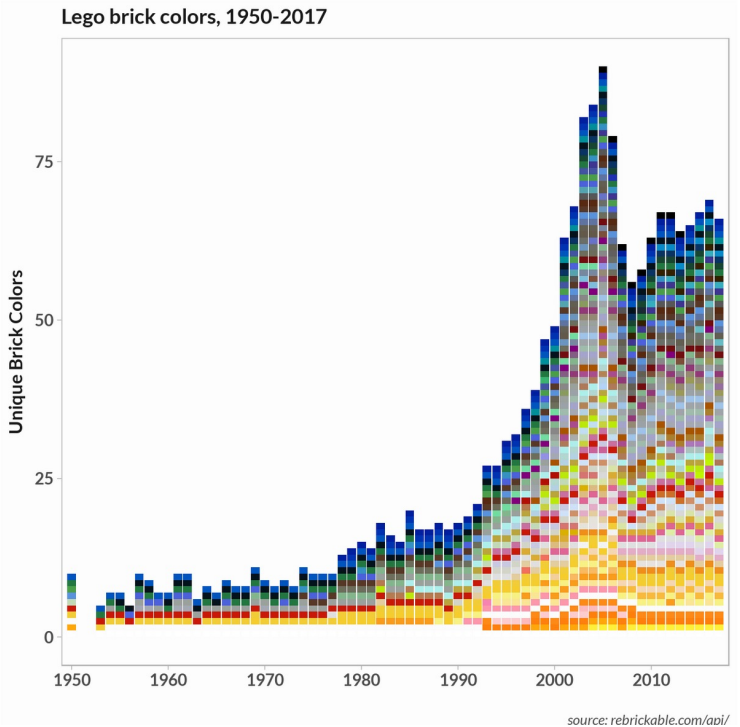


```
pyLDAvis.enable_notebook()
panel = pyLDAvis.sklearn.prepare(lda, tf_documents, tf_vectorizer, mds='tsne')
panel
```

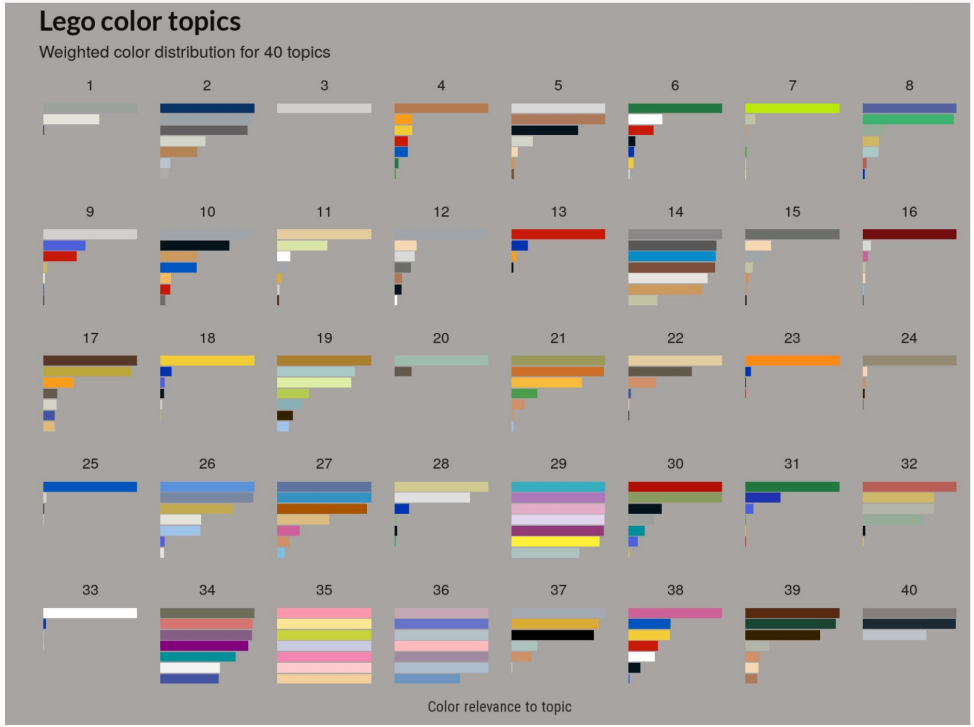


LEGO sets are documents, and LEGO bricks (by color) are words





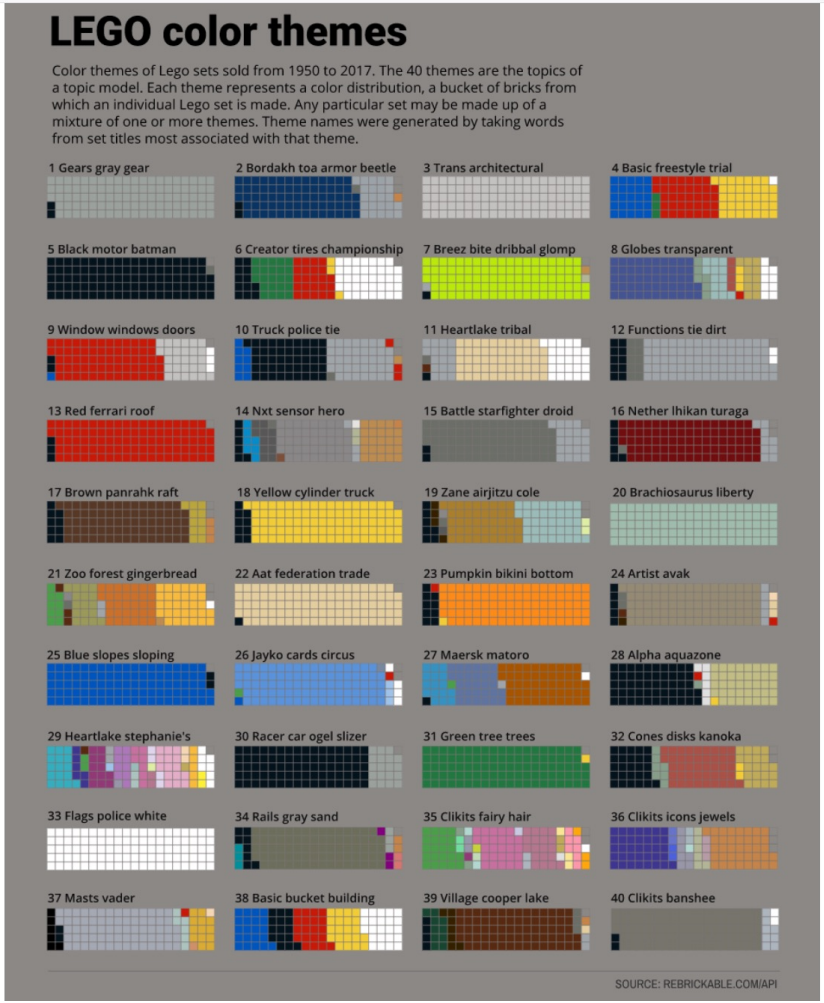
Source: https://nateaff.com/2017/09/11/lego-topic-models/



The above plot is based on the beta β matrix, which gives the posterior distribution of words given a topic, $p(w|t)$. The above plot shows a weighted β (or relevance) like that used in the `LDAvis` package.

$$\text{relevance}(w|t) = \lambda \cdot p(w|t) + (1-\lambda) \cdot \frac{p(w|t)}{p(w)}.$$

Source: https://nateaff.com/2017/09/11/lego-topic-models/



Source: https://nateaff.com/2017/09/11/lego-topic-models/

Pros and Cons of LDA

- Pros:
  - Useful as a visualization and exploration tool.
  - Reasonably efficient to estimate (but not as efficient as NMF).
  - Easy to modify & extend for specific scenarios
- Cons:
  - As with other topic modeling algorithms, humans must interpret the topics – but this can be difficult.
  - LDA is not specifically designed or trained to optimize tasks like classification.
  - The estimated topic vectors may not be good at discriminating documents.

General challenges with topic modeling

- How to interpret each topic?
  - Often topic modeling results in a few obvious topics, several that are clear after some thought, and the remaining 50% that are noise topics that look more random.
- How to generate a topic label for each?
  - What is the best exemplar term?
- How evaluate quality and effectiveness (relative or absolute)?
  - Data-based: What is the probability of a held-out dataset under the model?
  - Task-based, e.g. as features for supervised classification.

Turbotopics: visualizing topics with multi-word expressions [Blei and Lafferty 2009 https://arxiv.org/abs/0907.1013]

Comparison of first 4 topics found for Huffington Post news collection
by **standard LDA unigram topic model** (left) with **corresponding n-gram 'turbo topics'** (right)

film	obama	california	clinton
movie	barack	marriage	hillary
films	obamas	gay	clintons
movies	sen	court	campaign
hollywood	campaign	state	bill
documentary	senator	couples	shes
director	democratic	supreme	president
jones	illinois	decision	hillarys
screen	president	married	supporters
character	presidential	samesex	penn
cannes	recent	rights	politics
festival	political	marry	sexism
city	speech	law	political
theater	huffington	ruling	rodham
star	politics	states	democratic
hbo	michelle	equality	first
scene	voters	legal	say
actor	supporters	lesbian	sen
played	candidacy	equal	mrs
indiana	choice	appeals	presidency

movie	barack obama	marriage	hillary clinton
the film	obamas	state	campaign
hollywood	campaign	in california	bill clinton
director	sen barack obama	gay	shes
first	democratic	decision	the clinton
character	the illinois senator	court	hillarys
documentary	michelle	law	president
theater	recent	supreme court	sen clinton
best	speech	couples	mark penn
sex and the city	choice	ruling	politics
hbo	sen clinton	rights	sexism
scene	david axelrod	equality	the first
to make	president	legal	her campaign
release	camp	to marry	supporters
screen	the huffington post	married	made
actor	endorsed	samesex couples	fight
made	seen	states	called
stars	attacks	gay marriage	mrs clinton
indiana jones	political	sexual orientation	political
seen	gave	the california supreme court	hillary rodham clinton

The turbotopics algorithm is based on permutation testing and can be used with any topic model for which there is a latent topic assignment variable for each word of the corpus.

Evaluating topic models with perplexity and log-likelihood

- Requires a held-out test set: collection of unseen docs $\boldsymbol{w}_d$
- Topic model described by topic matrix Φ, hyperparameters $\boldsymbol{\alpha}$
- Higher log-likelihood : good

$$\mathcal{L}(\boldsymbol{w}) = \log p(\boldsymbol{w}|\boldsymbol{\Phi}, \alpha) = \sum_d \log p(\boldsymbol{w}_d|\boldsymbol{\Phi}, \alpha)$$

```
lda_model.score(data_vectorized)
```

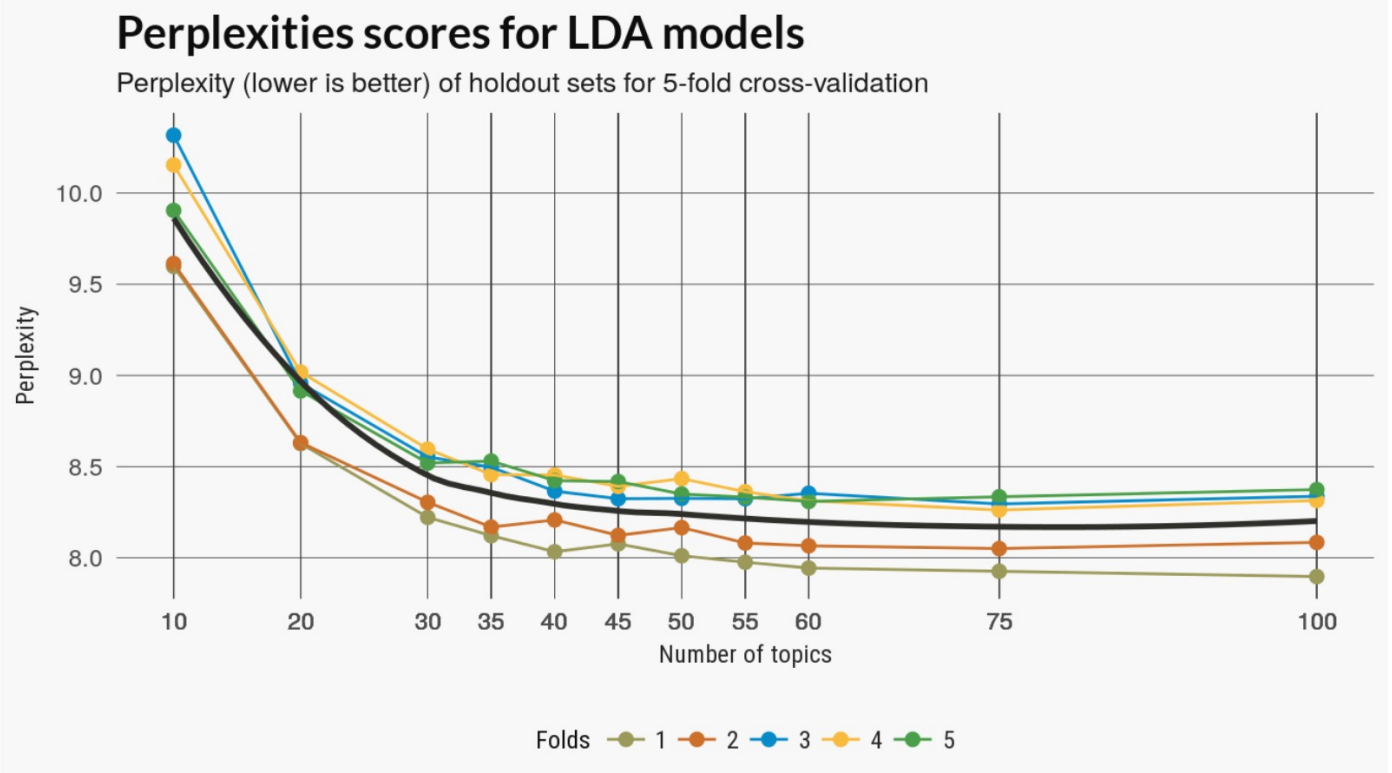
- Lower perplexity : good

$$\text{perplexity}(\text{test set } \boldsymbol{w}) = \exp\left\{-\frac{\mathcal{L}(\boldsymbol{w})}{\text{count of tokens}}\right\}$$

```
lda_model.perplexity(data_vectorized)
```

- Doesn't consider semantic associations or other set-quality attributes of the set of topic words/terms, e.g. coherence, redundancy.
- Problem: Perplexity not correlated with human judgment [Chang et al. 2009]
  http://www.cs.columbia.edu/~blei/papers/ChangBoyd-GraberWangGerrishBlei2009a.pdf

Using perplexity to choose the optimal number of topics *K*



Source: https://nateaff.com/2017/09/11/lego-topic-models/

Evaluating topic model quality with coherence measures

- Topic coherence: are the topic terms semantically related?
- Higher coherence is better
- Could also be a criterion that helps pick optimal # topics K

	Term
1	processor
2	memory
3	software
4	network
5	data

High coherence

	Term
1	football
2	airplane
3	software
4	agriculture
5	nuclear

Low coherence

Best practices with topic models

- Preprocessing
  - Stopwords & term filtering/weighting can have a major impact on the nature of the topics.
- Initialization and parameter settings
  - Like many optimization-based methods with complex solution spaces, different random initializations can lead to different results.
  - There may be several ‘reasonable’ choices for the number of topics K.

Best practices with topic models

- Scalability
  - Some other methods like NMF scale better than LDA.
  - But run time increases as the number of topics K increases.
- Interpreting Output
  - Consider further post-processing to reward solutions with more coherent topics.
  - Methods like turbotopics can make it easier to interpret topics.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

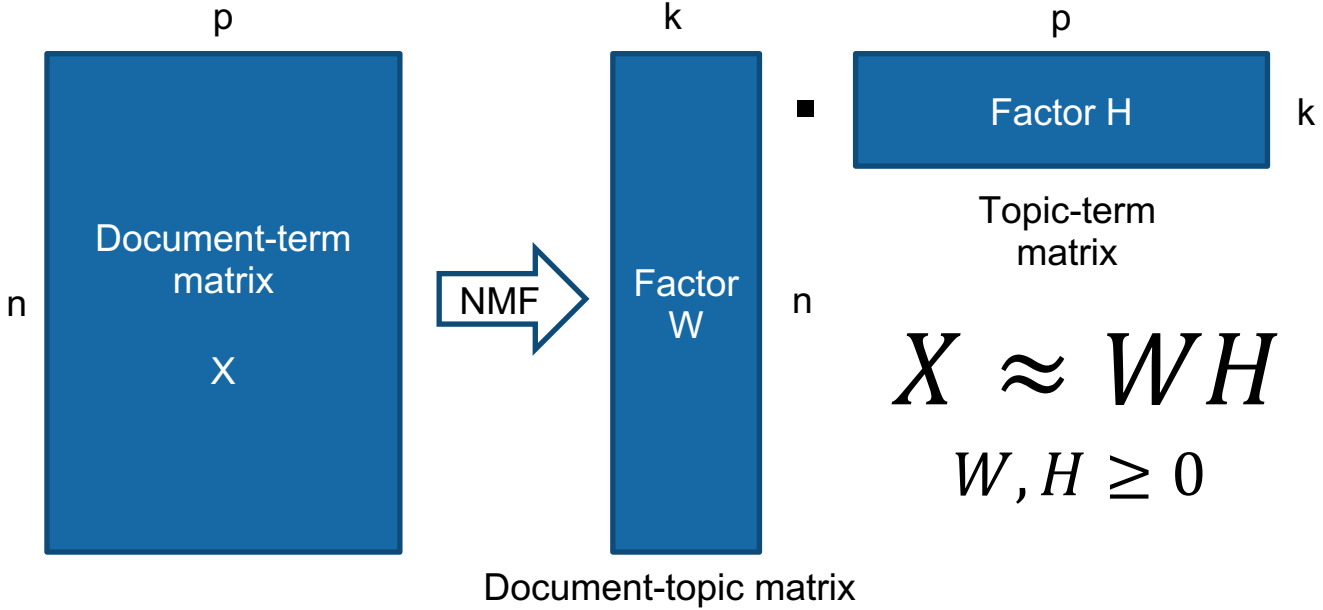
Topic Modeling 2: Non-negative Matrix Factorization

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Non-negative Matrix Factorization

- Non-negative Matrix Factorization (NMF) finds latent structure by factoring data as a non-negative matrix into two factors (Lee & Seung, 1999). It can be used for effective topic modeling.
- Input: (a) document-term matrix **X** of tf.idf values; (b) number of topics *k*
- Output: two k-dimensional factors **W** and **H** that when multiplied, approximate **X**



Original NMF paper: https://www.nature.com/articles/44565

A toy example

- Document corpus

```
documents_train = ['the cat and the dog and the duck were friends.',
                   'computers have power supplies.',
                   'plug in the monitor and turn on the computer.',
                   'you will find the plug on the right side of the screen.',
                   'my friend likes coffee and cats.',
                   'his dog gets along well with my friend.']
```

- Text preprocessing

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_documents = tfidf_vectorizer.fit_transform(documents_train)
tfidf_feature_names = tfidf_vectorizer.get_feature_names()
```

X =

(Sparse matrix)

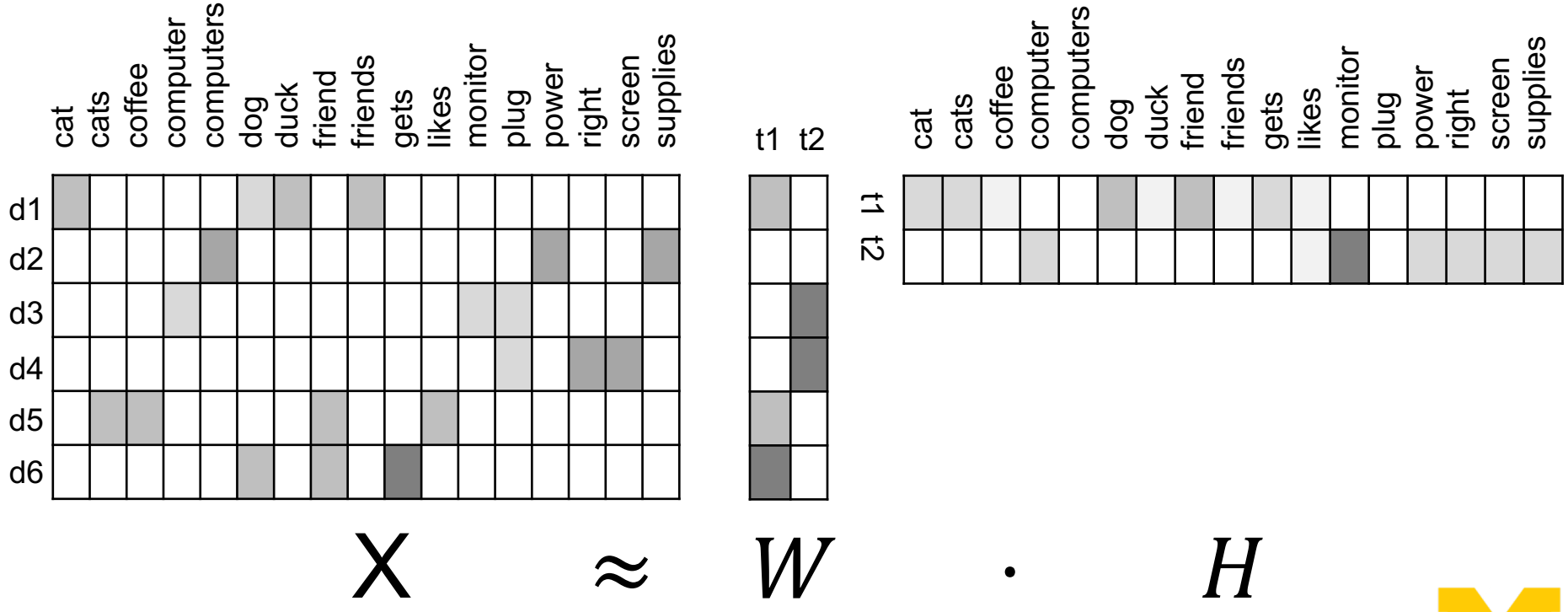
doc, term	tf.idf weight
(0, 8)	0.521
(0, 6)	0.521
(0, 5)	0.427
(0, 0)	0.521
(1, 16)	0.577
(1, 13)	0.577
(1, 4)	0.577
(2, 3)	0.521
(2, 17)	0.521
(2, 11)	0.521
(2, 12)	0.427
(3, 15)	0.611
(3, 14)	0.611
(3, 12)	0.501
(4, 1)	0.521
(4, 2)	0.521
(4, 10)	0.521
(4, 7)	0.427
(5, 9)	0.653
(5, 7)	0.535
(5, 5)	0.535

Vocabulary
(tfidf_feature_names)

0	cat
1	cats
2	coffee
3	computer
4	computers
5	dog
6	duck
7	friend
8	friends
9	gets
10	likes
11	monitor
12	plug
13	power
14	right
15	screen
16	supplies
17	turn

Factoring X into W and H with NMF

(k=2 topics)



Scikit-learn has an efficient NMF implementation

- The number of topics k (property: n_components) must be specified.
- Factors W and H are initialized to random values.
- With default NMF results can vary each time with the same input. So set the 'init' property to 'nndsvd' to get more stable results by initializing internally with SVD results. [Belford, MacNamee, Greene 2018]

```
from sklearn import decomposition

nmf = decomposition.NMF(n_components=n_topics, random_state=0, init="nndsvd")
W = nmf.fit_transform(X)
H = nmf.components_
```

The solver finds non-negative matrices *W, H* such that $||X - WH||^2$ is minimized w.r.t. *W* and *H.*
This is a non-convex problem to which the solver finds a local minimum using (complex) iterative update rules.

Belford, MacNamee, Greene 2018: http://derekgreene.com/papers/belford18eswa.pdf

Applying NMF in scikit-learn: Getting topic term weights from H

- The H factor has each topic's term weights
- Each row corresponds to a topic
- Each column corresponds to a unique term in the corpus vocabulary
- We can get the top-r terms for a topic by sorting that row's values and taking the top r.

```
for topic_index in range(0, topic_index_max):
    top_indices = np.argsort(H[topic_index, :])[::-1]
    top_terms = []
    for term_index in top_indices[0:top]:
        top_terms.append(tfidf_feature_names[term_index])
    print("topic ", topic_index, top_terms)
```

```
topic  0 ['friend', 'dog', 'gets', 'coffee', 'likes', 'cats']
topic  1 ['plug', 'screen', 'right', 'turn', 'monitor', 'computer']
```

Applying NMF in scikit-learn: Getting topic document weights from W

- The W factor holds document weights per topic.
- Each row corresponds to a document.
- Each columns corresponds to a topic.
- Sorting a topic's column gives a ranking of the most relevant documents for that topic.

```
top_docs = 2
for topic_index in range(0,2):
    top_indices = np.argsort(W[:, topic_index])[::-1]
    top_documents = []
    for doc_index in top_indices[0:top_docs]:
        doc = documents_train[doc_index]
        print(topic_index, doc_index, doc)
        top_documents.append(doc)
```

```
0 5 his dog gets along well with my friend.
0 4 my friend likes coffee and cats.
1 3 you will find the plug on the right side of the screen.
1 2 plug in the monitor and turn on the computer.
```

topic index

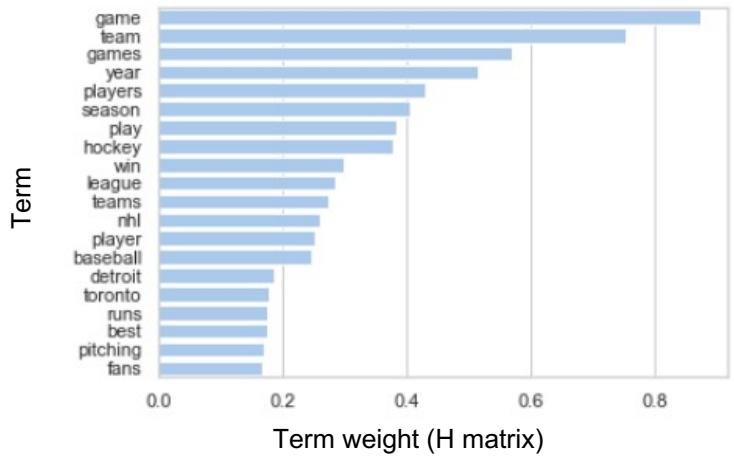
doc index

For longer documents we could use the document's title or search result snippet.

Applying NMF in scikit-learn: the 20 newsgroups dataset

Topics (K=10). Highlighting the ‘sports’ topic #8

```
topic  0 ['don', 'just', 'like', 'think', 'know', 'good', 've', 'time']
topic  1 ['windows', 'file', 'dos', 'window', 'files', 'program', 'use', 'using']
topic  2 ['geb', 'dsl', 'n3jxp', 'chastity', 'cadre', 'pitt', 'shameful', 'intellect']
topic  3 ['god', 'jesus', 'bible', 'believe', 'christ', 'faith', 'christian', 'christians']
topic  4 ['key', 'chip', 'encryption', 'clipper', 'keys', 'government', 'escrow', 'use']
topic  5 ['drive', 'scsi', 'card', 'disk', 'drives', 'controller', 'ide', 'hard']
topic  6 ['thanks', 'does', 'know', 'mail', 'advance', 'info', 'hi', 'anybody']
topic  7 ['00', 'edu', 'new', 'space', 'sale', 'com', '10', 'price']
topic  8 ['game', 'team', 'games', 'year', 'players', 'season', 'play', 'hockey']
topic  9 ['people', 'israel', 'government', 'armenian', 'jews', 'israeli', 'state', 'armenians']
```



Top documents per topic

```
0 1533 **************  This is somewhat long, but pleas read it!!!
0 6437 Accounts of Anti-Armenian Human Rights Violations in Azerbai

1 2391 I quit windows normally to run a special DOS app, got done w
1 3451 Hi there, I'm having a bizarre video problem within Windows

2 8550 So just what was it you wanted to say?
2 8660 By law, they would not be allowed to do that anyhow.

3 5559 I have come across what I consider to be an excellent tract.
3 6962  excellent question timothy. i hpoe the answers you get will

4 5898 Here is a revised version of my summary which corrects some
4 9365 The following document summarizes the Clipper Chip, how it i

5 3725 I have a 486sx25 computer with a 105 Mg Seagate IDE drive an
5 7722 : >>point of view, why does SCSI have an advantage when it c

6 8571 Hi, Does anyone know anything about this group and what the
6 10616 Thanks for the info.  I assume that this is for MFC 1.0.  Do

7 4730 Comics for sale. All are Marvel and the majority of the comi
7 7432 Miscellaneous comics for sale. I really would like to get ri

8 9635 Archive-name: hockey-faq rec.sport.hockey answers to Freque
8 3366 Philadelphia at Chicago:  Teams tied for 1st after Sunday

9 70 : Pardon me? Here is to an amherst-clown:: : "Your three
9 3754 Center for Policy Research writes...Your comparison
```

Comparing LDA and NMF topics

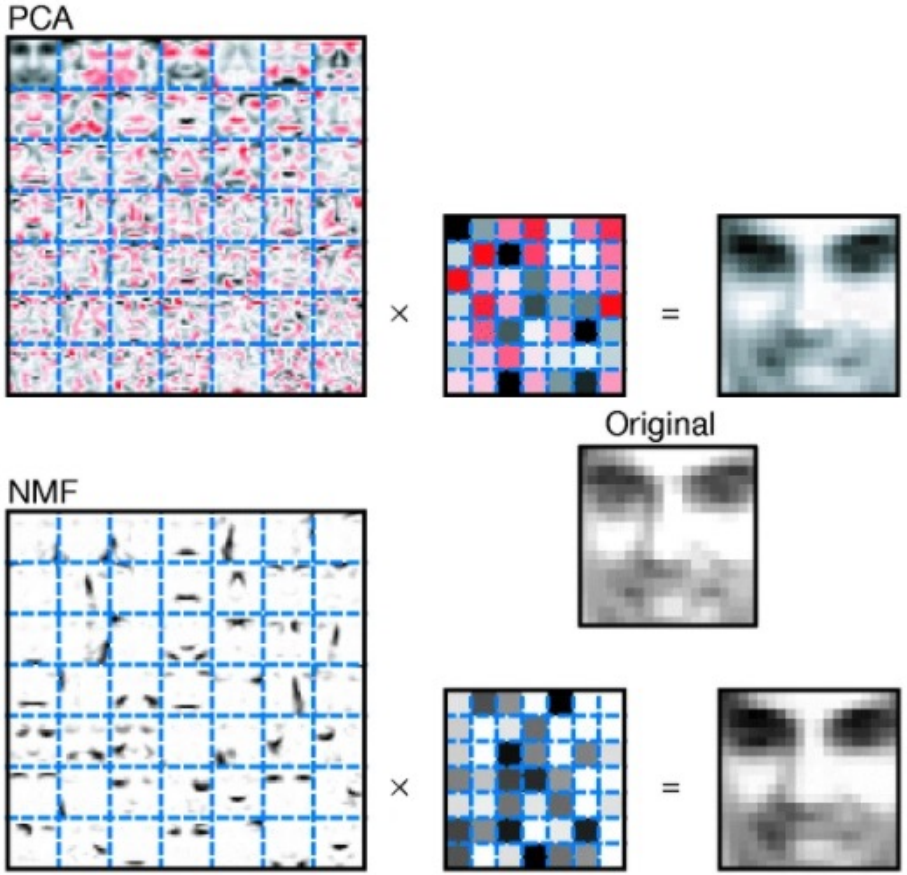
20newgroups dataset

NMF

```
topic  0 ['don', 'just', 'like', 'think', 'know', 'good', 've', 'time']
topic  1 ['windows', 'file', 'dos', 'window', 'files', 'program', 'use', 'using']
topic  2 ['geb', 'dsl', 'n3jxp', 'chastity', 'cadre', 'pitt', 'shameful', 'intellect']
topic  3 ['god', 'jesus', 'bible', 'believe', 'christ', 'faith', 'christian', 'christians']
topic  4 ['key', 'chip', 'encryption', 'clipper', 'keys', 'government', 'escrow', 'use']
topic  5 ['drive', 'scsi', 'card', 'disk', 'drives', 'controller', 'ide', 'hard']
topic  6 ['thanks', 'does', 'know', 'mail', 'advance', 'info', 'hi', 'anybody']
topic  7 ['00', 'edu', 'new', 'space', 'sale', 'com', '10', 'price']
topic  8 ['game', 'team', 'games', 'year', 'players', 'season', 'play', 'hockey']
topic  9 ['people', 'israel', 'government', 'armenian', 'jews', 'israeli', 'state', 'armenians']
```

LDA

```
topic 0: ['use', 'like', 'know', 'used', 'power', 'drive', 'new', 'don']
topic 1: ['just', 'like', 'don', 'good', 'think', 'time', 'car', 've']
topic 2: ['god', 'people', 'don', 'think', 'does', 'just', 'say', 'jesus']
topic 3: ['ax', 'max', 'g9v', 'b8f', 'a86', 'pl', '145', '1d9']
topic 4: ['people', 'government', 'mr', 'gun', 'president', 'don', 'think', 'know']
topic 5: ['edu', 'windows', 'use', 'file', 'com', 'available', 'software', 'program']
topic 6: ['space', 'file', 'university', 'program', 'nasa', '1993', 'israel', 'research']
topic 7: ['key', 'scsi', 'drive', 'chip', 'encryption', 'use', 'clipper', 'keys']
topic 8: ['game', 'team', 'year', 'db', 'armenian', 'said', 'people', 'armenians']
topic 9: ['00', '10', '25', '17', '11', '15', '16', '14']
```



Comparing the factorization of PCA and NMF

Source: (Lee & Seung, 1999)

PCA learns a holistic (global) representation.
NMF learns a parts-based representation.

Why? These differences arise from the differences in constraints on matrix factors.

PCA constrains W to be orthonormal, rows of H to be orthogonal to each other. W and H can be positive or negative.

NMF constrains W and H to be positive. Only additive elements are allowed. Parts combine to form the whole.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

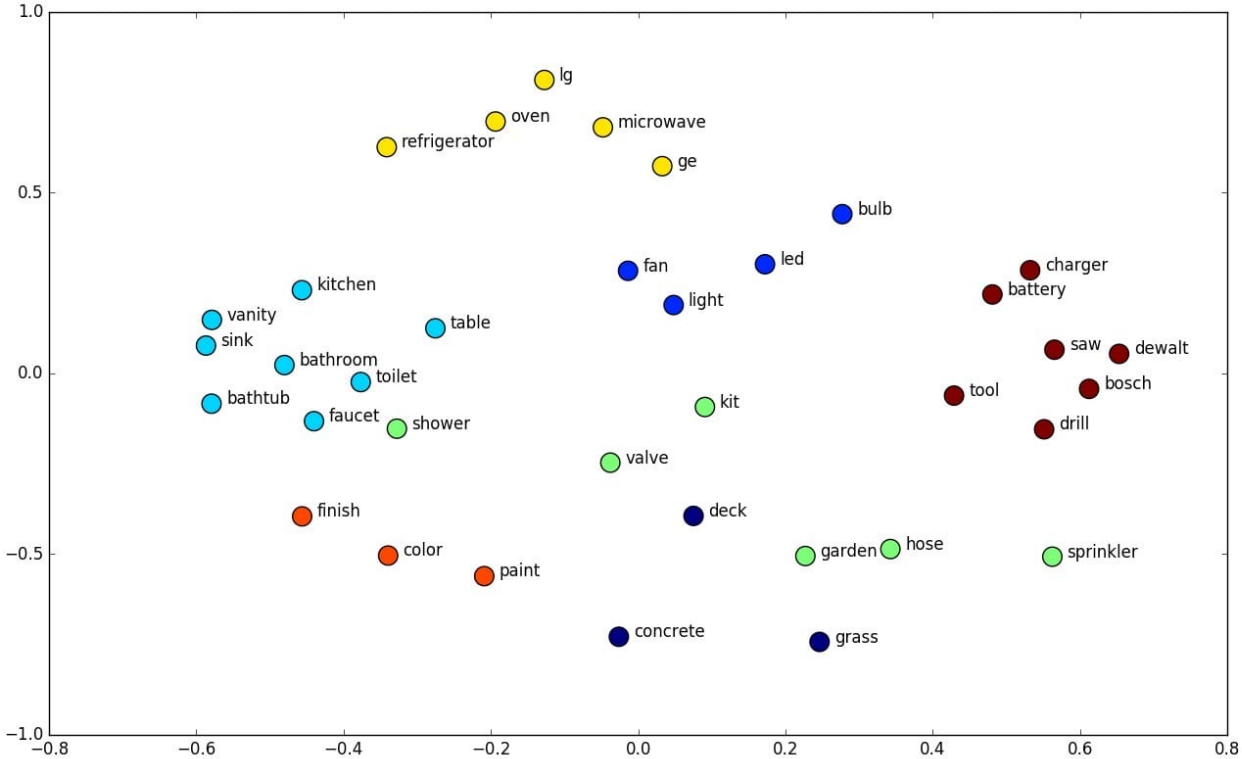
Word Embeddings

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

A word embedding maps a term to a vector

- "A word is known by the company it keeps." – John R. Firth (1957)
- Distributional hypothesis: we can deduce a word's meaning by how it's actually used in language.
- Embeddings have been associated recently with deep learning methods. But we don't need DL methods to compute useful word embeddings…



Source: https://easyai.tech/en/ai-definition/word-embedding/

Sparse word vectors: 1-hot high-dimensional encodings

- No semantic notion of similarity
- Even synonyms have different 1-hot encodings.
- Would be nice if similar words had similar numeric vectors.
- Recall the “vocabulary gap” problem of Latent Semantic Indexing.

Why is it useful to represent a word/term with a numeric vector?

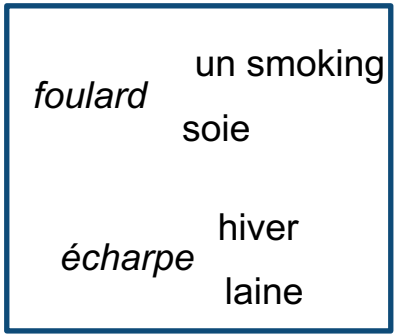
- Useful anywhere we need to represent a word's semantics (meaning).
- The concept of word embeddings is central to Natural Language Processing.
- NLP tasks can exploit models of semantic distance between words.

- Word similarity and semantic matching
- Semantic cohesion (see topic modeling)
- Stemming and inflections
- Machine translation
- Part-of-Speech tagging
- Named entity recognition
- Relationship extraction
- Analogy solving
- And more…

Example:

How to translate 'scarf' to French?
Should it be *foulard* or *écharpe* ?

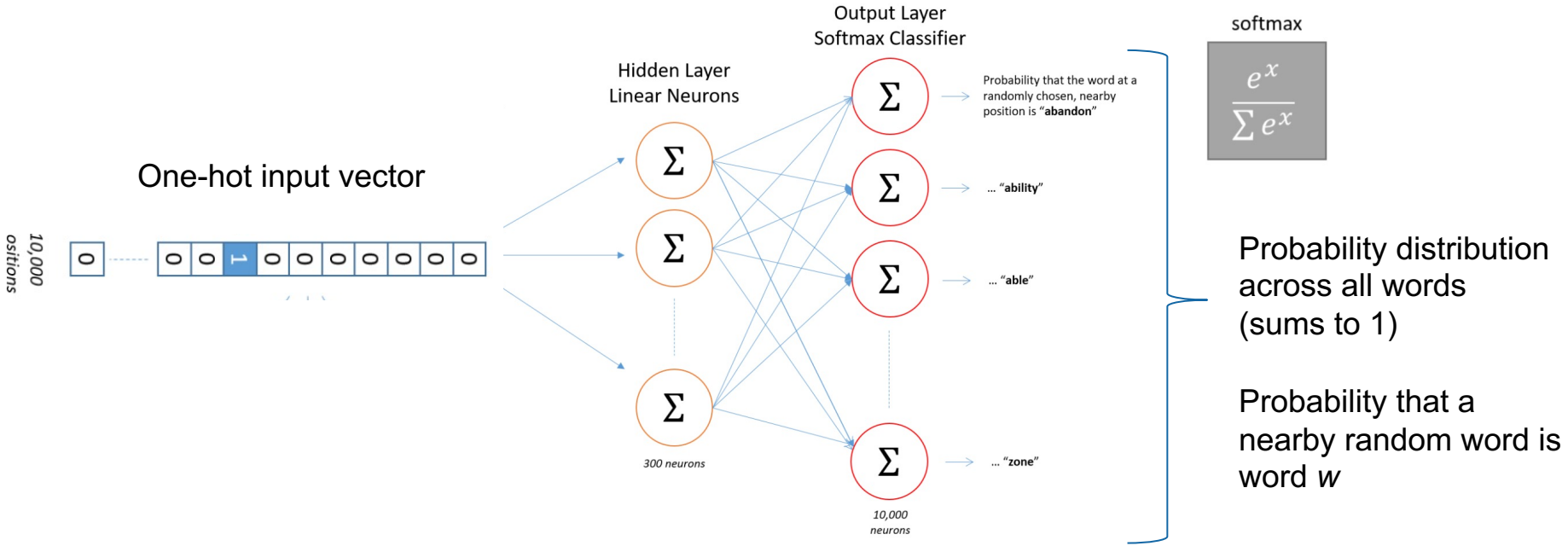
The colors of his silk scarf peeking above his collar looked elegantly paired with his tuxedo.



Two examples of trained word embeddings

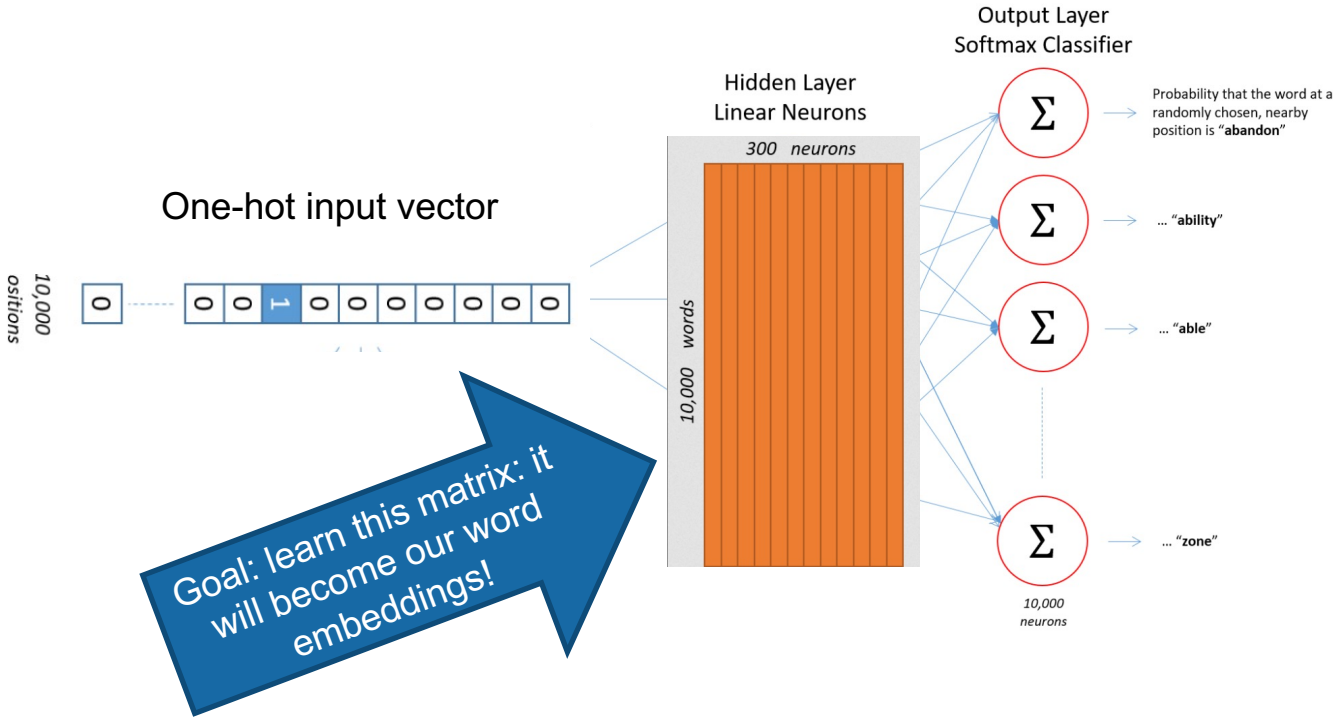
- Word2vec (local context)
- GloVE (local and global context)

word2vec: skip-gram model



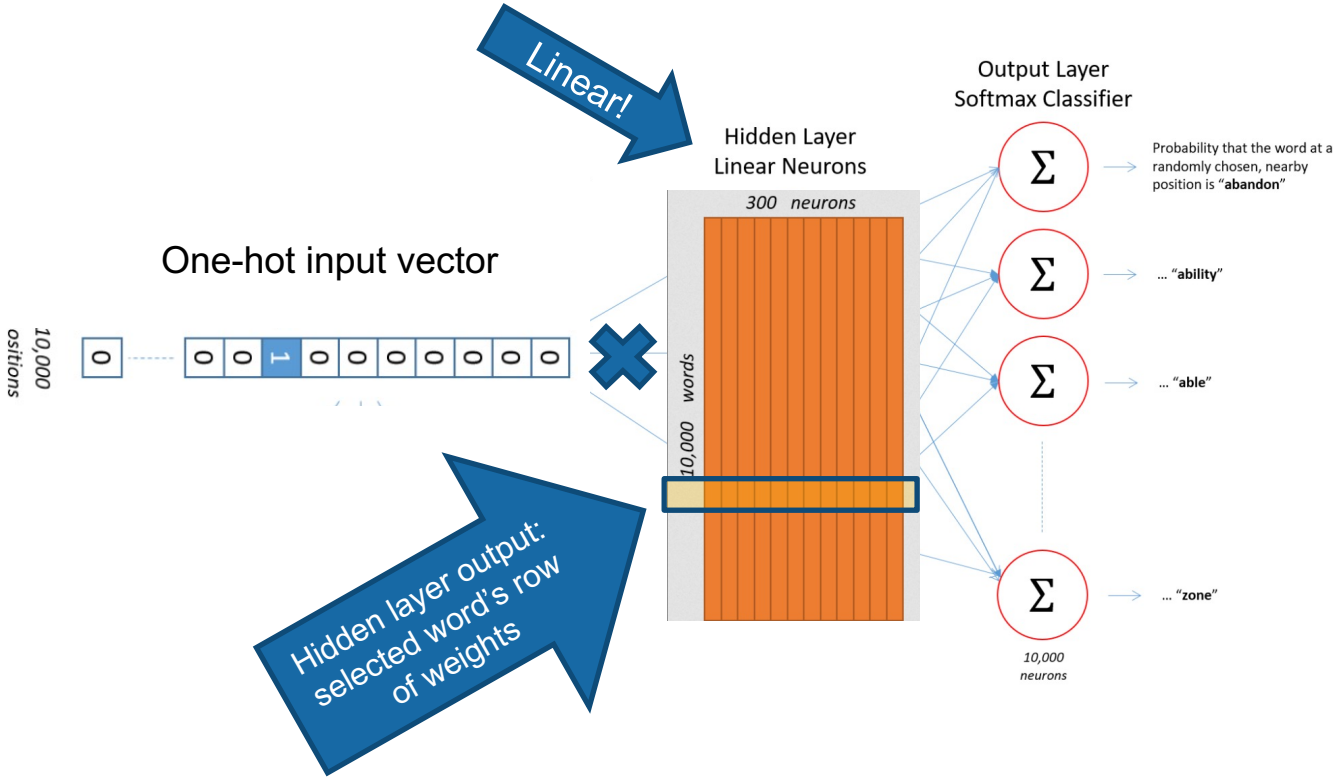
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec: skip-gram model



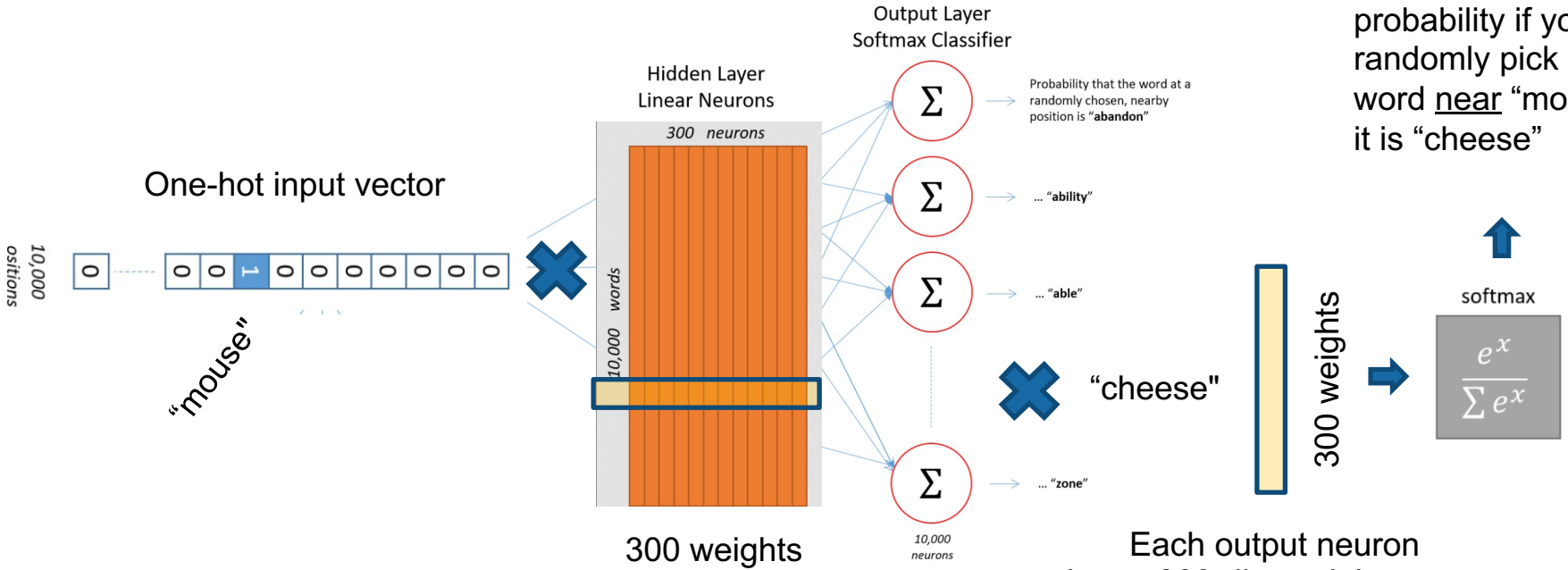
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec: skip-gram model



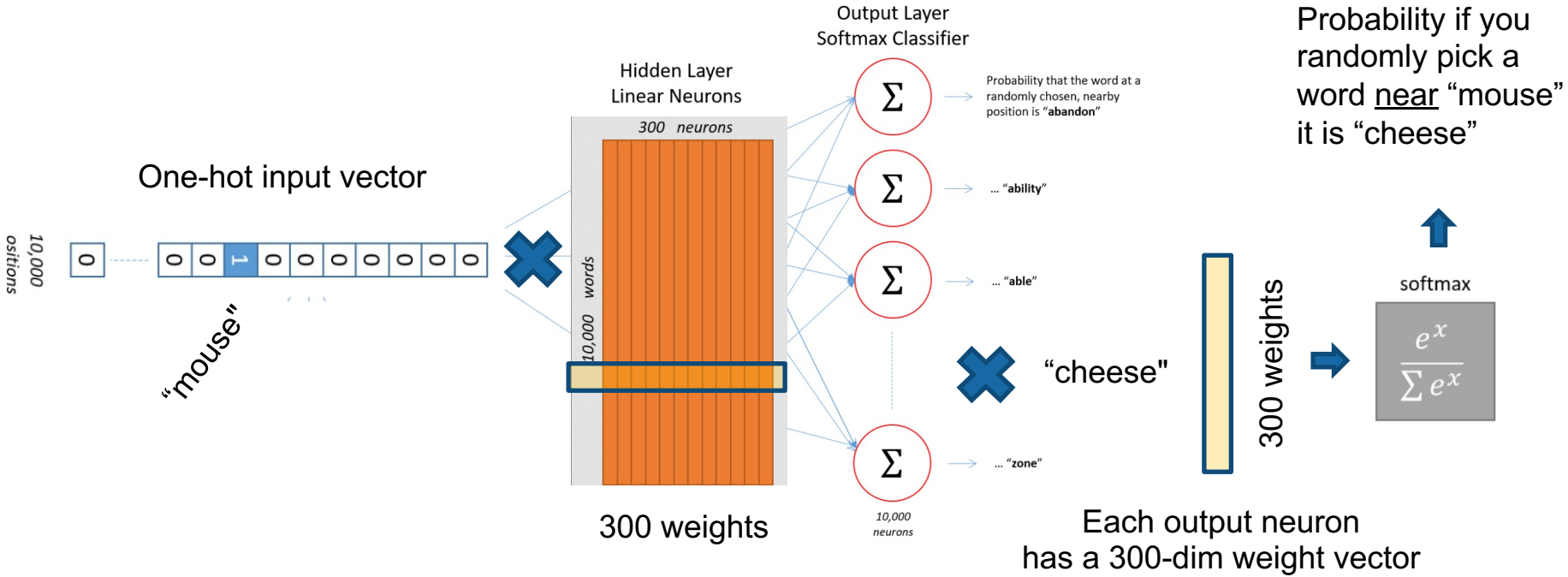
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec: skip-gram model



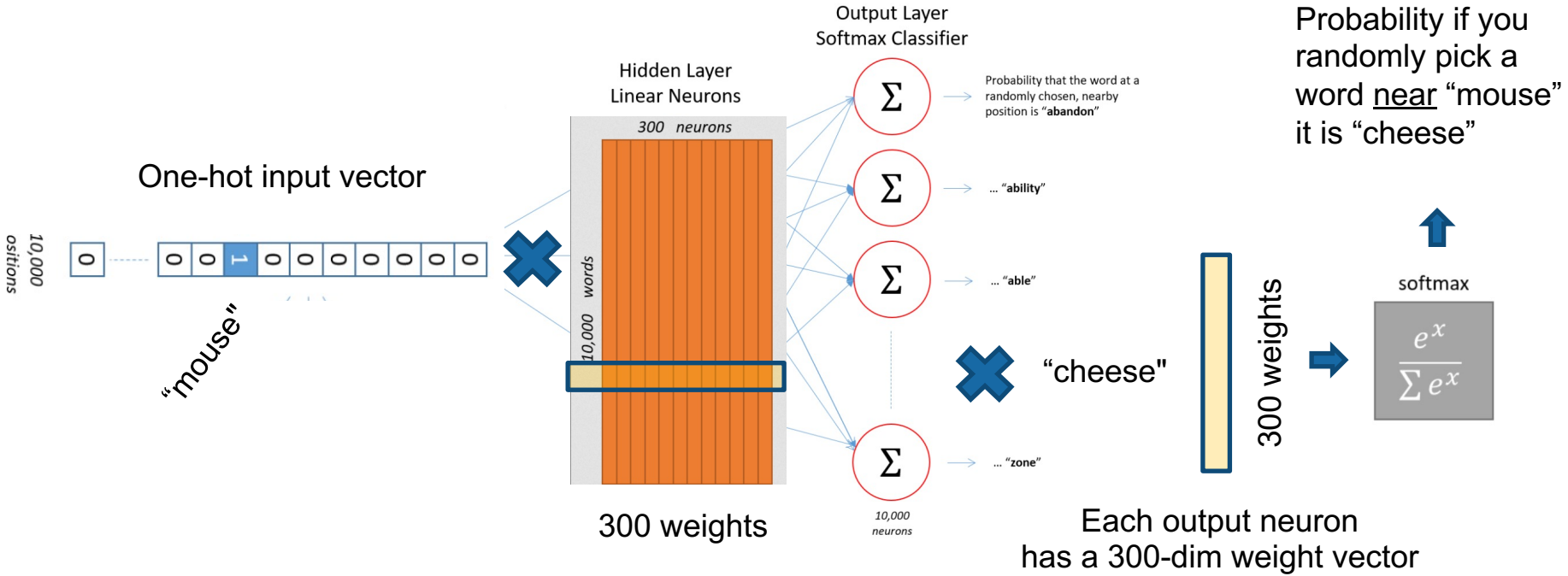
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

The model should learn to output similar context predictions for words used in similar ways…how could it do this?



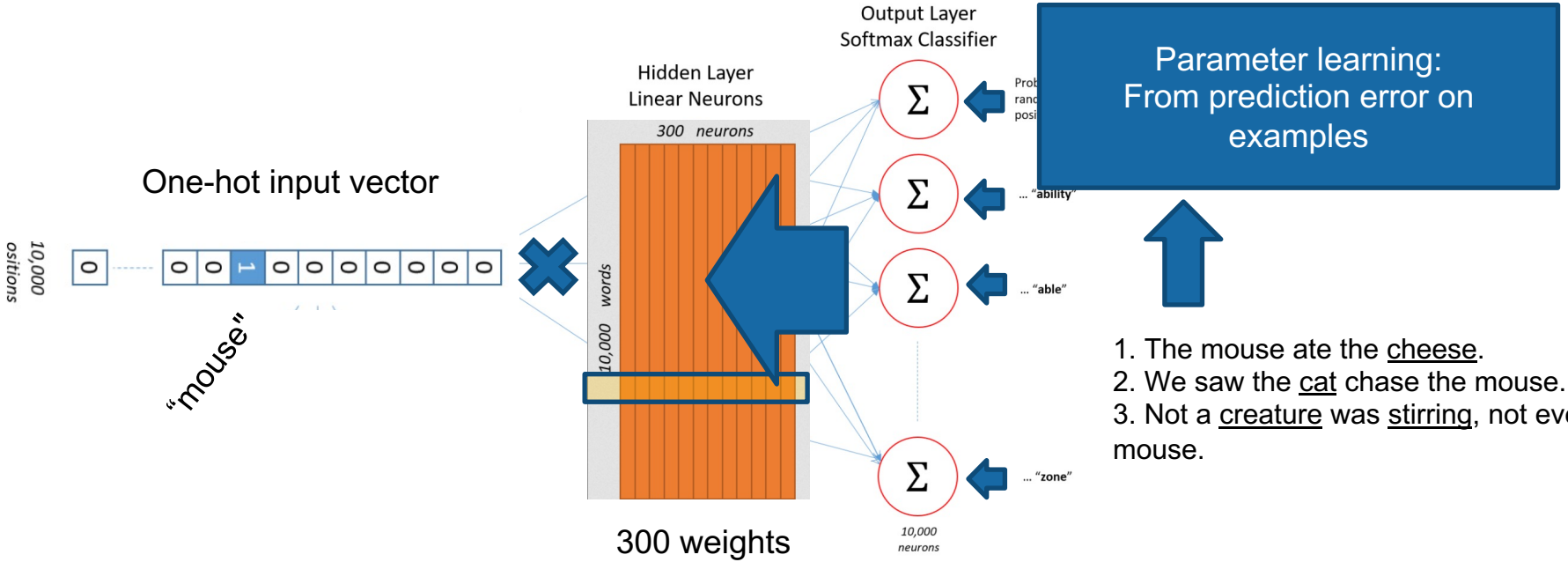
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

It should learn similar hidden layer weights for the two contextually-similar words!



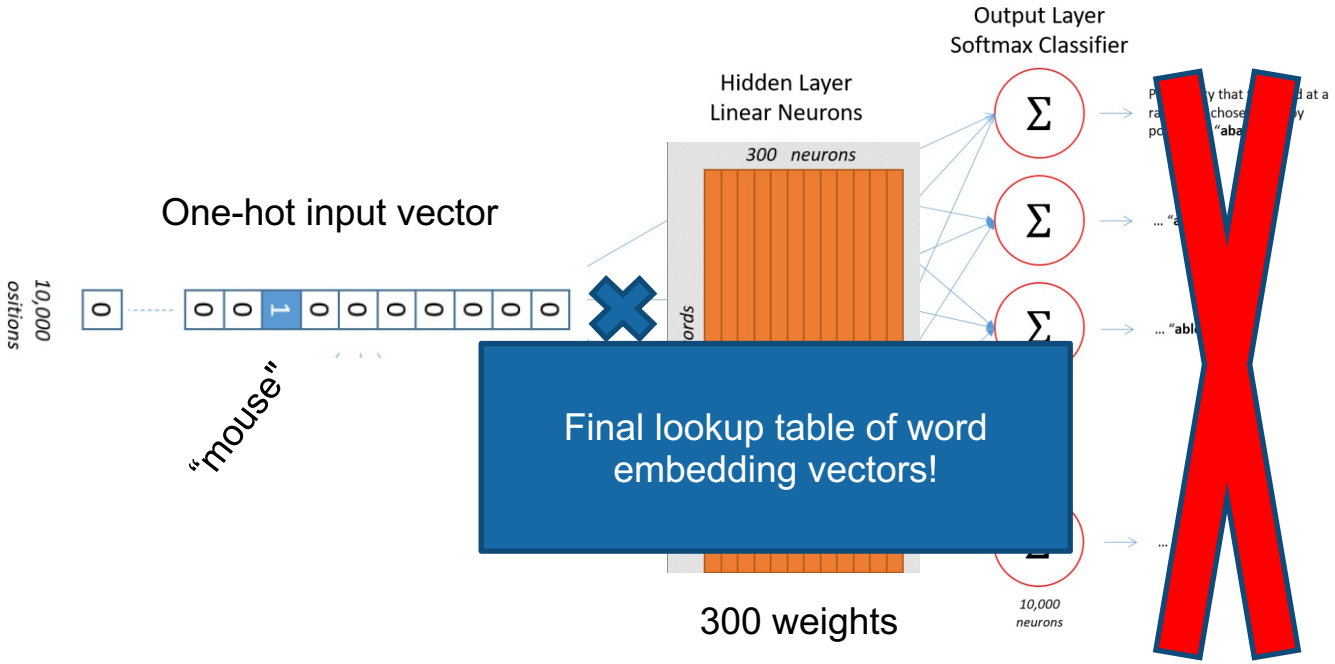
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

The hidden layer weights are learned via a parameter learning algorithm (backpropagation, negative sampling, …)



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

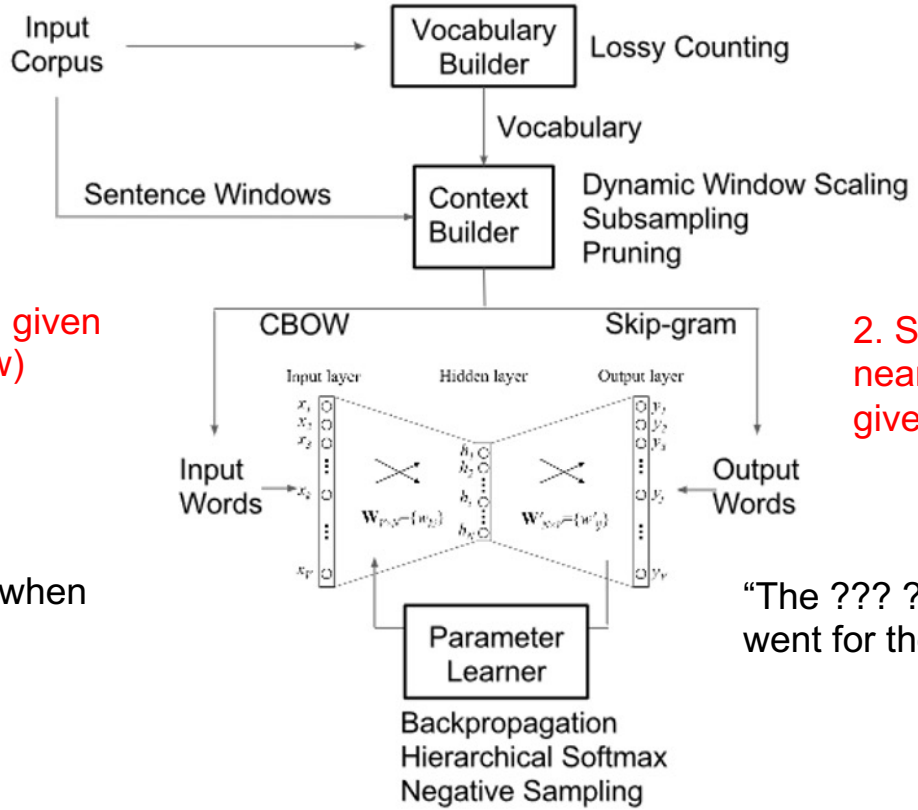
The hidden layer weights are learned via a parameter learning algorithm (backpropagation, negative sampling, …)



Surprisingly, we don't actually care about the prediction task! Just the representation (embedding) that it leads to being created.
This is an example of a self-supervised learning "pretext" task (see week 4).

Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec has two variants: skip-gram and CBOW



1. CBOW: Predict target word given nearby words (context window)

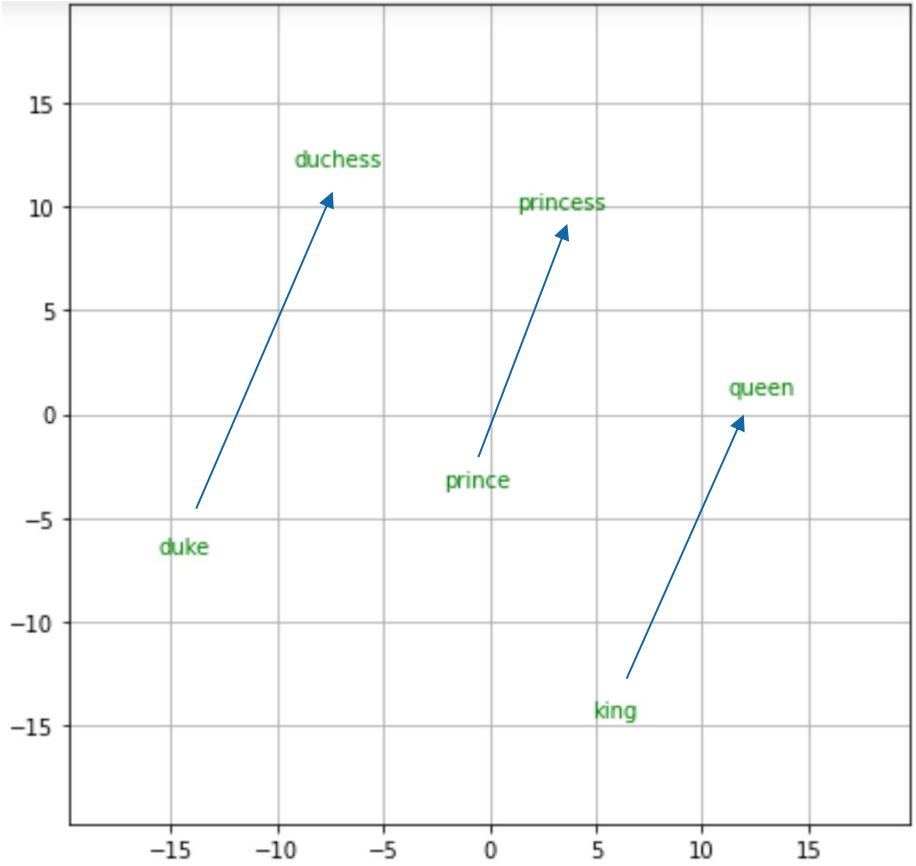
2. Skip-gram: predict nearby words given target word

"The cat caught the _____ when it went for the cheese.

"The ??? ??? the mouse when it went for the ???.

Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

The linear semantics of neural word embeddings



king + (woman - man) = queen

Suppose Δ = woman – man

Then the following also hold for these embedding vectors:

duke + Δ = duchess

prince + Δ = princess

GloVE (short for “Global Vectors”)

- word2vec captured local context within a neighborhood of a target word.
- GloVE attempts to extend word2vec by incorporating global co-occurrence information into the embeddings.
- It learns word relationships from the word co-occurrence statistics of any (large enough) corpus.
- Optimizes the embeddings such that the cosine similarity between words reflects the number of times they co-occur, which helps make the resulting vectors more interpretable.

Let `xij` be the number of times words `i` and `j` “co-occur” in a given corpus, and `xi` be the number of times the word `i` occurred in general. Provided the corpus is large enough, we can consider that the probability that a word `i` occurs next to the word `j` is:

$$\mathbb{P}(j|i) = \frac{X_{ij}}{X_i}$$

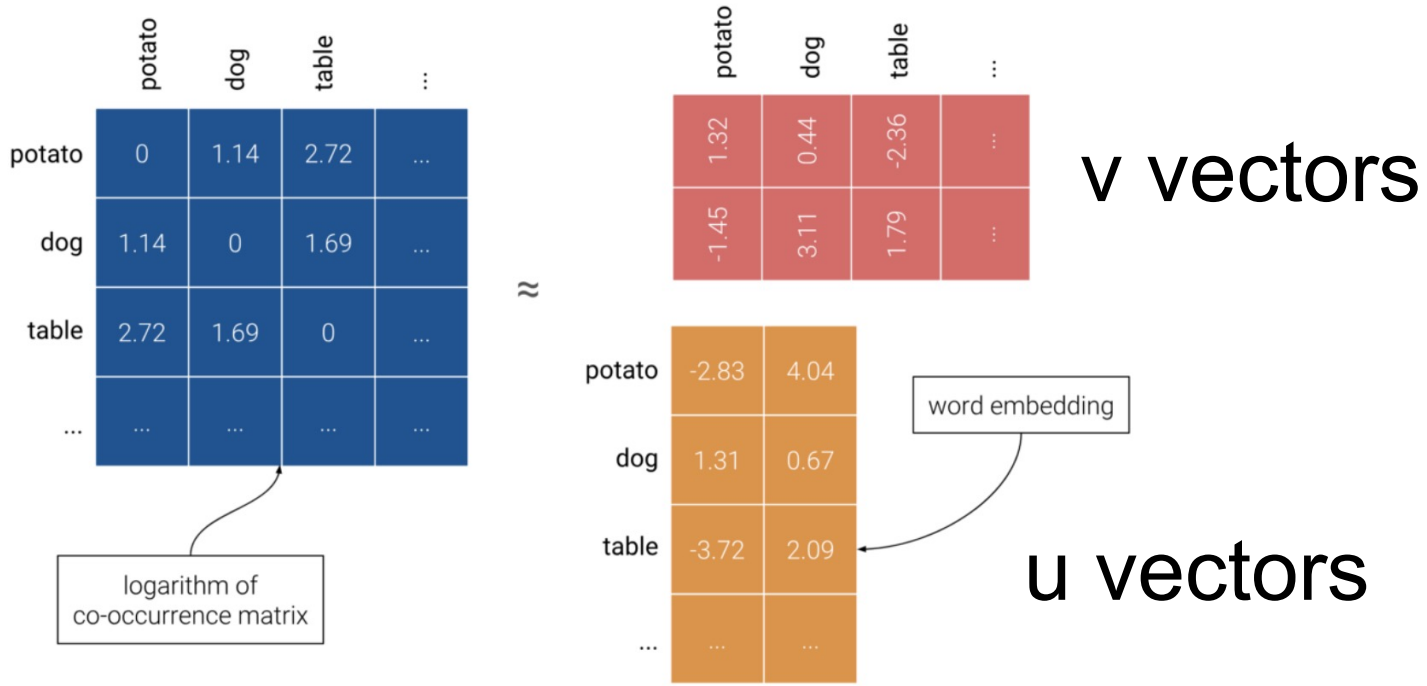
Source: https://nlp.stanford.edu/projects/glove/

GloVE main idea: Use the ratios of co-occurrence probabilities, rather than the co-occurrence probabilities themselves.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Source: https://nlp.stanford.edu/projects/glove/

GloVE is matrix factorization of the cooccurrence matrix



Source: https://medium.com/data-from-the-trenches/arithmetic-properties-of-word-embeddings-e918e3fda2ac

GloVE is a log-bilinear model with a least-squares objective (minimizing approximation error of the cooccurrence matrix)

$$\mathbf{u}_i^T \mathbf{v}_k + b_i^u + b_k^v = \log[X_{ik}]$$

$$\sum_{i,j=1}^{V} f(X_{ij}) \left(\mathbf{u}_i^T \mathbf{v}_k + b_i^u + b_k^v - \log[X_{ik}] \right)^2$$

For every possible pair of words i, j

X_ij = cooccurrence count of i, j
u = word embedding vector
v = context vector

b = bias terms

Strengths and weaknesses of popular embedding algorithms

Word Embedding Method	Strengths	Weaknesses
word2vec: skip-gram	- Effective with small training sets and low-frequency terms	- Slow training
word2vec: CBOW	- Significantly faster, better accuracy for more common terms	- Less effective for rare words
GloVe	- Easier to interpret - More stable results than word2vec	- Needs lots of memory to store word co-occurrence stats

Stability results: https://laura-burdick.github.io/papers/naacl18embeddings.pdf

Python word embedding implementations

- **Gensim** (library for topic modeling and NLP)
  1. scikit-learn wrapper: **W2VTransformer** object (used in this course)
  2. see models.word2vec docs at https://radimrehurek.com/gensim/models/word2vec.html

- Original word2vec C code by **Google**
  https://code.google.com/archive/p/word2vec/

- The Zeugma library has an implementation of GloVE and other embedding types

https://github.com/nkthiebaut/zeugma

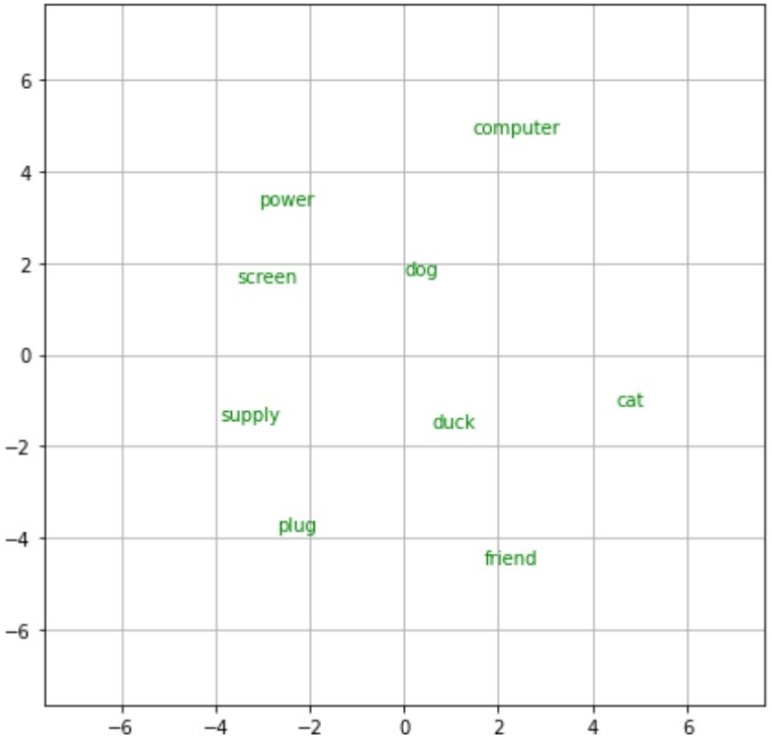
```
from zeugma.embeddings import EmbeddingTransformer

glove = EmbeddingTransformer('glove')
x_train = glove.transform(corpus_train)

model = LogisticRegression()
model.fit(x_train, y_train)

x_test = glove.transform(corpus_test)
model.predict(x_test)
```

```
from gensim.sklearn_api import W2VTransformer

# Create a model to represent each word by a 10 dimensional vector.
model = W2VTransformer(size=10, min_count=1, seed=2)

# Parse the silly documents into a form W2VTransformer wants:
# Each document is a list of strings (words)
simple_corpus = []
for i in range(len(simple_documents_train)):
    t = simple_documents_train[i].lower().split(' ')
    simple_corpus.append(t)

# Now fit the word2vec model with the corpus, and use the fit model
# to embed the given  words into a 10-dimensional space
embed_words = ['cat', 'dog', 'duck', 'supply', 'friend', 'power', 'computer', 'plug', 'screen']
wordvecs = model.fit(simple_corpus).transform(embed_words)


def dump_word_embeddings(words, wordvecs):
    for i in range(0, len(words)):
        print(words[i], "\t", wordvecs[i])
```

```
dump_word_embeddings(embed_words, wordvecs)
```

```
cat      [ 0.01107736 -0.01789192  0.00865422  0.04715009  0.02875026  0.01151348
  0.03989485  0.04727173 -0.00552519 -0.02450414]
dog      [ 0.01211248  0.02945776 -0.03043586 -0.01122049 -0.04830866  0.00036707
 -0.01327686  0.0047334  -0.00272162 -0.02590703]
duck     [-0.02280007  0.0150022   0.04723163 -0.00906663  0.00134643 -0.0119813
  0.02129506  0.03737264 -0.04527682 -0.01945323]
supply   [ 0.04057245  0.04225577  0.02304681 -0.01670146 -0.03275366 -0.03825971
  0.01745637  0.00197733 -0.00667339  0.02740229]
friend   [ 0.03540802  0.0075277   0.02727438  0.0169314  -0.03166131  0.03125474
  0.04037235  0.02600932 -0.04072101  0.04918338]
power    [ 0.01410934  0.04800265  0.00097991 -0.02503773  0.01859954 -0.00589897
 -0.03646366 -0.00096323  0.03608064 -0.02543611]
computer         [-0.04694019  0.03090378 -0.04720047 -0.02375052  0.0120469   0.00807231
  0.04437716 -0.04357917 -0.00541033 -0.04409727]
plug     [ 0.04929583  0.04033271  0.02988797  0.04780739  0.00807764 -0.0295534
 -0.02616698  0.01661818 -0.03995521  0.01179908]
screen   [-0.02053462  0.03736921  0.03091021 -0.03546154  0.00551345 -0.03377008
 -0.00692282  0.0119453   0.01761552 -0.04064646]
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS

X_embed_normalized = StandardScaler().fit(wordvecs).transform(wordvecs)

mds = MDS(n_components = 2, random_state = 0)
X_embed_mds = mds.fit_transform(X_embed_normalized)
plot_word_embedding_vectors(X_embed_mds, np.transpose(embed_words))
```



Embeddings for higher-level representations

- doc2vec: paragraphs
- sent2vec: sentences
- dna2vec: DNA/RNA
- tweet2vec: hashtag prediction
- item2vec: product suggestions
- graph2vec: networks

Tweets	Word model baseline	*tweet2vec*
ninety-one degrees.*❤😎	#initialsofsomeone.. #nw #gameofthrones	#summer **#loveit** #sun
self-cooked scramble egg. yum!! !url	#music #cheap #cute	**#yummy** #food #foodporn
can't sleeeeeeep	#gameofthrones #heartbreaker	#tired **#insomnia**
oklahoma!!!!!!!!!! champions!!!!!	#initialsofsomeone.. #nw #lrt	**#wcws** #sooners #ou
7 % of battery . iphones die too quick .	#help #power #money #s	#fml #apple #bbl **#thestruggle**
i have the cutest nephew in the world !url	#nephew **#cute** #family	#socute **#cute** #puppy

#hashtags predicted by tweet2vec

Source: https://arxiv.org/abs/1605.03481

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information