Some Advanced Techniques for Modeling Text

Outline

- **In content-based recommendation,**
  - The feature vector of text is very high and sparse.
  - E.g., considering the size of vocabulary.

- **What do we need:**
  - Low-dimensional and dense vector of text.
  - Better for text similarity computation.

- **What we cover --- Dimension reduction for text:**
  - Principle Component Analysis (PCA).
  - Probabilistic topic models (for text).
    - Probabilistic Latent Semantic Analysis.
    - Latent Dirichlet Allocation.
  - Distributional representations of words.

PCA

What is PCA?

- **General idea:**
  - Performing eigendecomposition of the covariance matrix for a given datasets X.
  - Can reduce the dimension of feature vectors.

- **Basic procedures:**
  - Data: $X \in R^{n \times m}$, with n dimension, m examples, .

1. For each row of X, calculating its mean value.
2. For each value in a row of X, subtracting its mean value corresponding to the row.
3. Computing the covariance matrix, $C = \frac{1}{\mathrm{m}} X X^T$.
4. Computing the eigenvalues and eigenvectors of C.
5. Sorting the eigenvectors by the eigenvalues. Assuming we only keep k eigenvectors at the top, we can get the transformation matrix P.
6. Finally, getting the dimension-reduced data $Y = PX$.

A simple example

	x	y
	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
Data =	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9

	x	y
	0.69	0.49
	-1.31	-1.21
	0.39	0.99
	0.09	0.29
DataAdjust =	1.29	1.09
	0.49	0.79
	0.19	-0.31
	-0.81	-0.81
	-0.31	-0.31
	-0.71	-1.01

Covariance

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

A simple example

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

Eigenvalue and eigenvector

$$\Big\downarrow$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

A simple example--transformation

	x	y
	0.69	0.49
	-1.31	-1.21
	0.39	0.99
	0.09	0.29
DataAdjust =	1.29	1.09
	0.49	0.79
	0.19	-0.31
	-0.81	-0.81
	-0.31	-0.31
	-0.71	-1.01

Transformed Data (Single rigenvector)
x
-0.827970186
1.77758033
-0.992197494
-0.274210416
-1.67580142
-0.912949103
0.991094375
1.14457216
0.438046137
1.22382056

Eigenvalue: 1.28402771
Eigenvector:
(0.677873399, -0.735178656)

The law behind PCA

- **Space transformation**
  - Inner product
    - Given two vectors, the inner product is,
    - One form:

$$(a_1, a_2, \cdots, a_n) \cdot (b_1, b_2, \cdots, b_n)^\mathsf{T} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

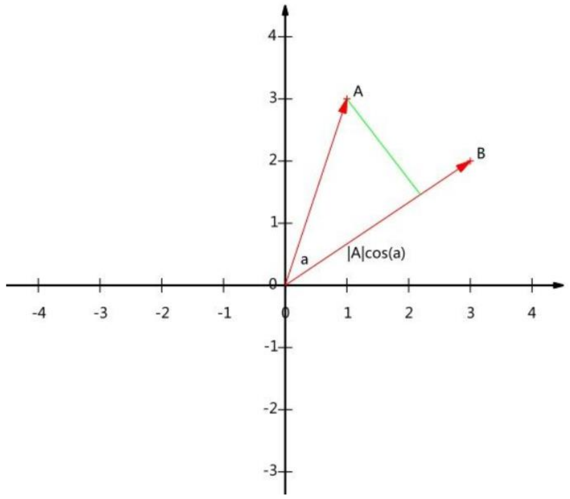
    - Another form:

$$A = (x_1, y_1), \quad B = (x_2, y_2) \; A \cdot B = |A||B|cos(\alpha)$$

    - If |B|= 1,

$$A \cdot B = |A|cos(a)$$

    - Thus, it could be regarded as the projection of A to B.



The law behind PCA

- **Space transformation**
  - Expanding to multiple vectors,
    - $B = [b_{11}, b_{12}, \dots;\ b_{21}, b_{22}, \dots; \dots;]$
    - $a = [a_1, a_2, \dots]$
  - Ba -> projecting vector a to the space formed by B.
    - $c_1 = [b_{11}, b_{12}, \dots] \cdot [a_1, a_2, \dots]^T$
    - …
    - $c_k = [b_{k1}, b_{k2}, \dots] \cdot [a_1, a_2, \dots]^T$
  - $[c_1, \dots, c_k]$ forms a new vector of a.
  - When k is small, we can achieve the dimension reduction.

- **Now, the question becomes how to determine the transformation matrix B.**

The law behind PCA

- **One perspective: maximum variance formulation**
  - Suppose $k = 1$, meaning that the transformed space only has 1 dimension.
  - We have a collection of data points $\{\boldsymbol{x}_n\}$, $n = 1, \ldots , N$.
    - Each $\boldsymbol{x}_n \in R^D$.
    - Mean, $\overline{\boldsymbol{x}}=\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n$
  - The variance of the projected data is given by

$$\max \frac{1}{N}\sum_{n=1}^{N}({\boldsymbol{b}_1}^T\boldsymbol{x}_n - {\boldsymbol{b}_1}^T\overline{\boldsymbol{x}})^2 = {\boldsymbol{b}_1}^T\boldsymbol{S}\boldsymbol{b}_1$$

  - S corresponds to the data covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^{\mathrm{T}}$$

    - This explains why we perform the minus operation in PCA.

The law behind PCA

- **Maximal Optimization**

$$\max \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{b}_1{}^T\boldsymbol{x}_n - \boldsymbol{b}_1{}^T\overline{\boldsymbol{x}})^2 = \boldsymbol{b}_1{}^T\boldsymbol{S}\boldsymbol{b}_1$$

  - Adding a constraint to b_1,
    - $\boldsymbol{b}_1{}^T\boldsymbol{b}_1 = 1.$
  - Lagrange multiplier method,

$$\max \boldsymbol{b}_1{}^T\boldsymbol{S}\boldsymbol{b}_1 + \gamma_1(1 - \boldsymbol{b}_1{}^T\boldsymbol{b}_1)$$

  - Derivative equal to 0,

$$\boldsymbol{S}\boldsymbol{b}_1 = \gamma_1\boldsymbol{b}_1$$

  - Left-multiply $\boldsymbol{b}_1{}^T$,

$$\boldsymbol{b}_1{}^T\boldsymbol{S}\boldsymbol{b}_1 = \gamma_1$$

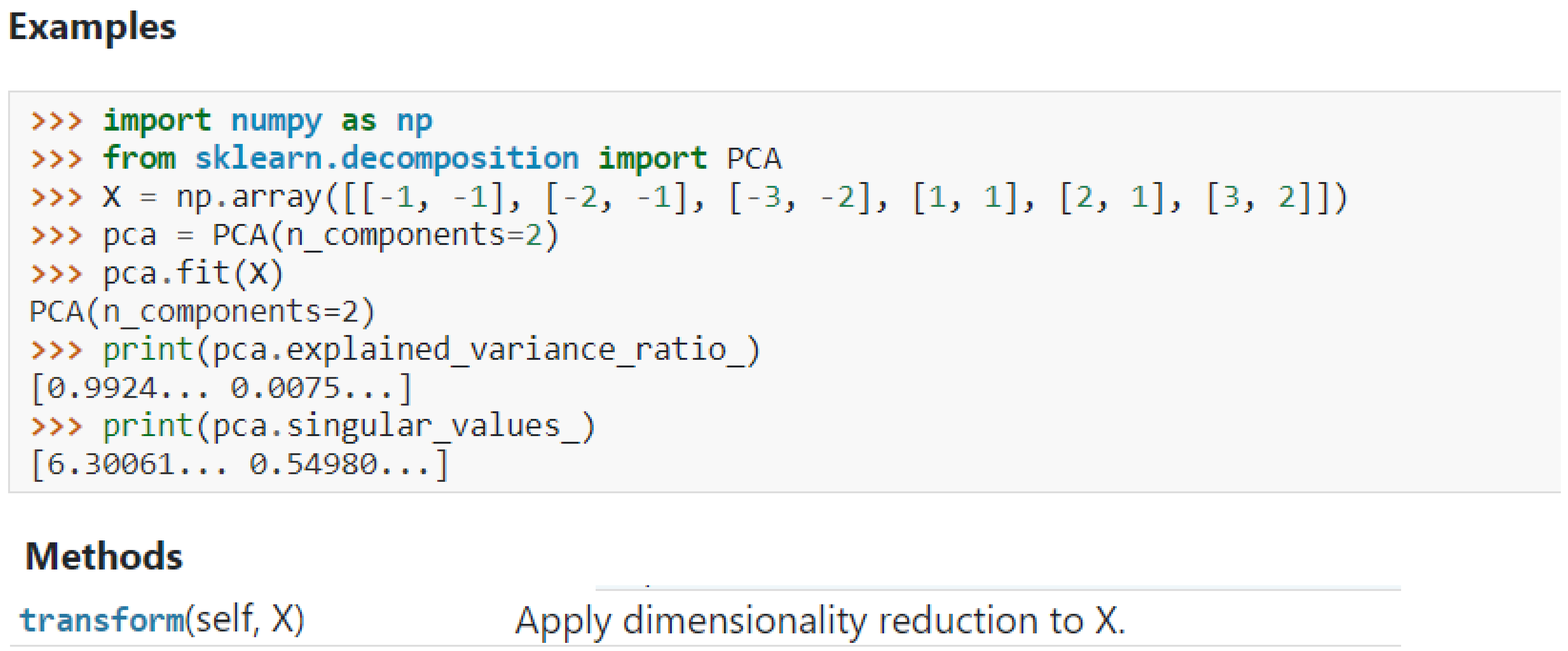
    - Recall our target is to maximize $\boldsymbol{b}_1{}^T\boldsymbol{S}\boldsymbol{b}_1$, thus γ_1 is what we want.

The law behind PCA

- **By now, we have gotten the largest eigenvalue and its eigenvector ($\boldsymbol{b_1}$).**
- **How do we extend it to multiple dimensions ($\boldsymbol{k > 1}$).**
  - Iteratively choose the largest eigenvalues among the left ones.
  - Their eigenvectors could be used for projection.

References for PCA

- **API for using PCA**
  - Scikit-learn:
    - https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA



Examples

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
>>> print(pca.singular_values_)
[6.30061... 0.54980...]
```

Methods

`transform`(self, X)	Apply dimensionality reduction to X.

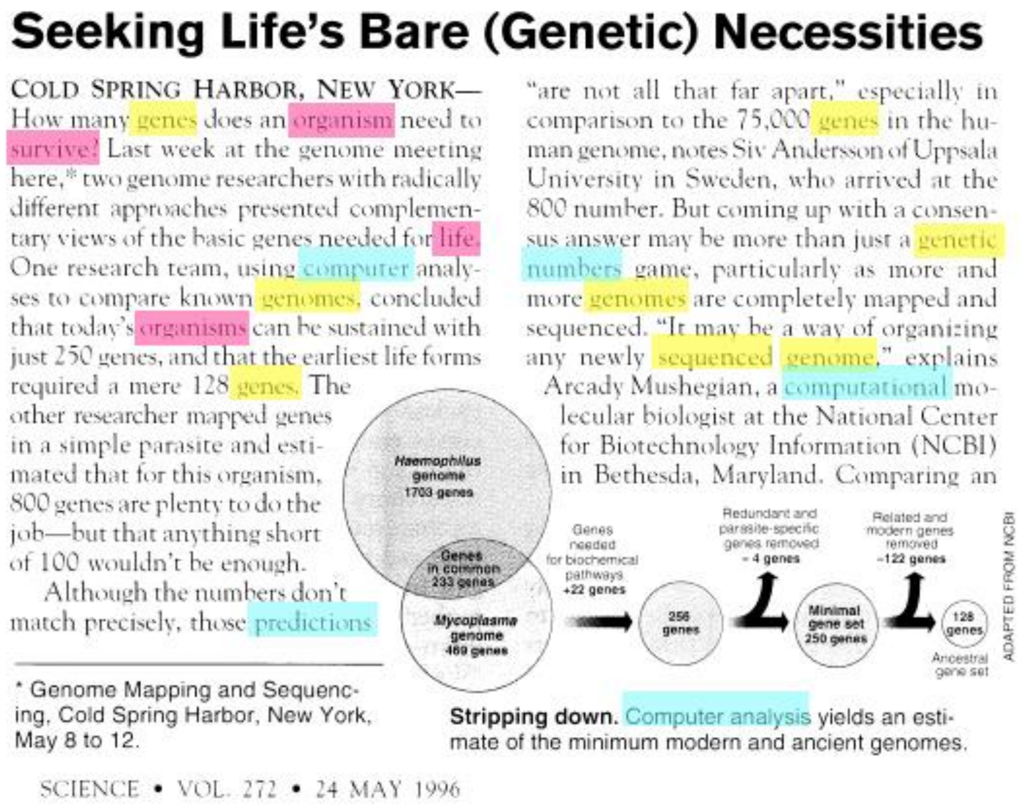
Probabilistic Topic Models

Text Modeling

- **Motivation**
  - Suppose you’re given a massive corpora and asked to carry out the following tasks
    - Organize the documents into themantic categories
    - Enable a domain expert to analyze and understand the content
    - Find relationships between the categories

- **General concept**
  - A method of (usually unsupervised) discovery of latent or hidden structure in a corpus
    - Applied primarily to text corpora, but techniques are more general

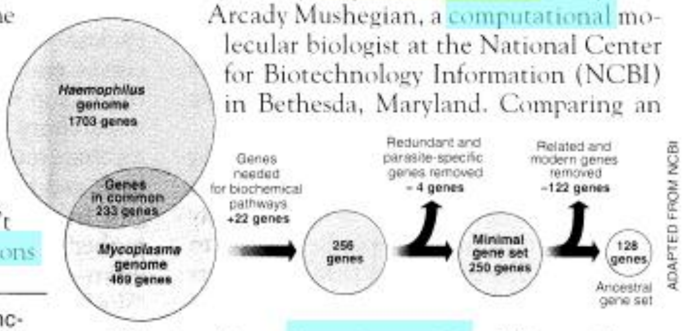
Intuition Behind Topic Model



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an



Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed -4 genes
Minimal gene set 250 genes
Related and modern genes removed -122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

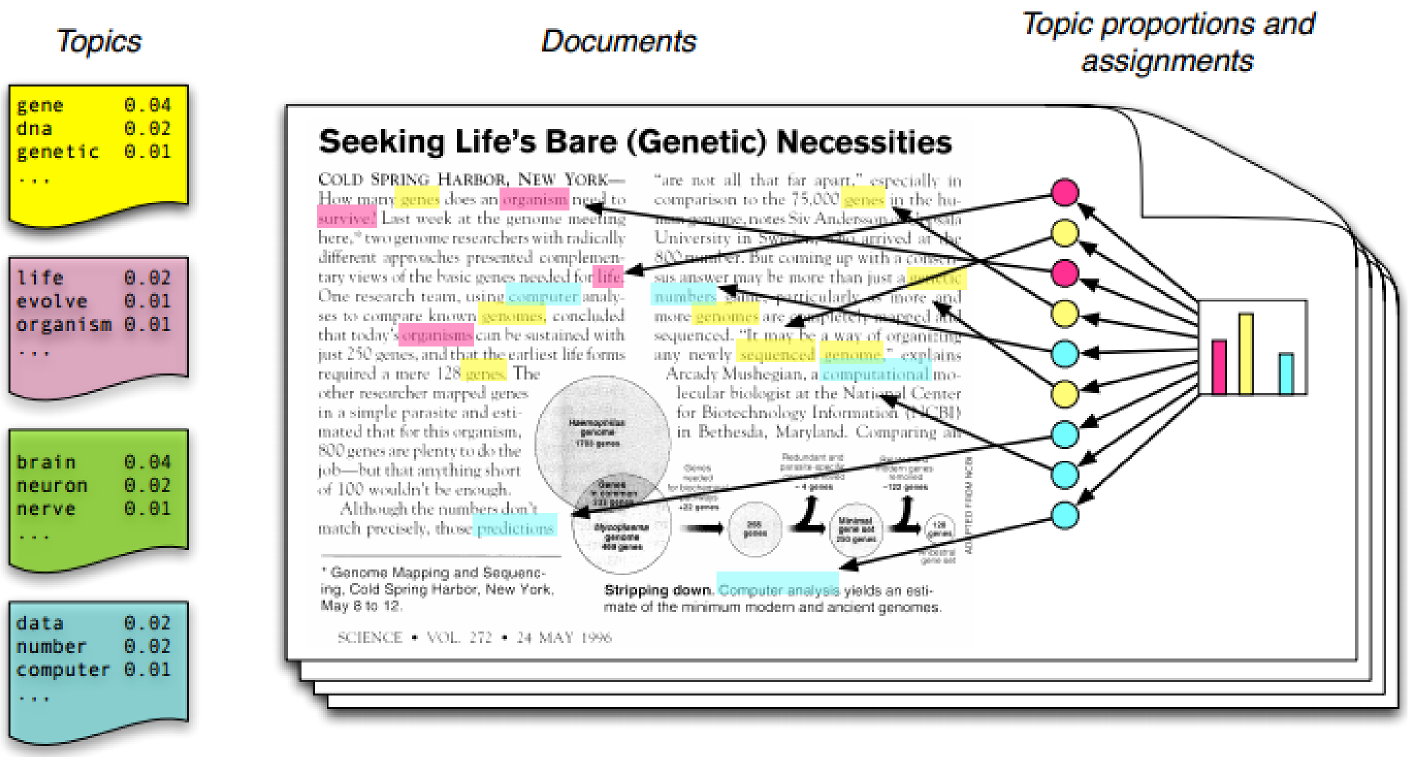
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

Simple intuition: Documents exhibit multiple topics.

Generative Model



- Each document is a random mixture of corpus-wide topics
- Each word is drawn from one of those topics

Probabilistic Latent Semantic Analysis (PLSA)[1]

- **一篇文档(Document) 可以由多个主题(Topic) 混合而成， 而每个Topic 都是词汇上的概率分布，文章中的每个词都是由一个Topic 生成的。在 PLSA 模型中，上帝是按照如下的游戏规则来生成文本的：**

1: 上帝有两种类型的骰子，一类是doc-topic 骰子,每个doc-topic 骰子有K 个面，每个面是一个topic 的编号；一类是topic-word 骰子，每个topic-word 骰子有V 个面，每个面对应一个词；

doc-topic

topic-word

2: 上帝一共有K 个topic-word 骰子, 每个骰子有一个编号，编号从1 到K；

3: 生成每篇文档之前，上帝都先为这篇文章制造一个特定的doc-topic 骰子，然后重复如下过程生成文档中的词

- 投掷这个doc-topic 骰子,得到一个topic 编号z
- 选择K 个topic-word 骰子中编号为z的那个，投掷这个骰子，于是得到一个词

[1]Probabilistic Latent Semantic Analysis, (UAI, 1999)

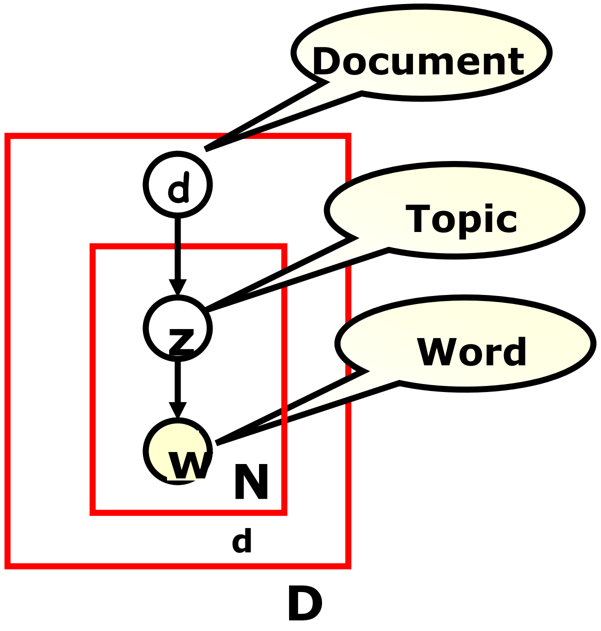
Probabilistic Latent Semantic Analysis (PLSA)

- 于是在 PLSA 这个模型中，第*m*篇文档 *dm* 中的每个词的生成概率为

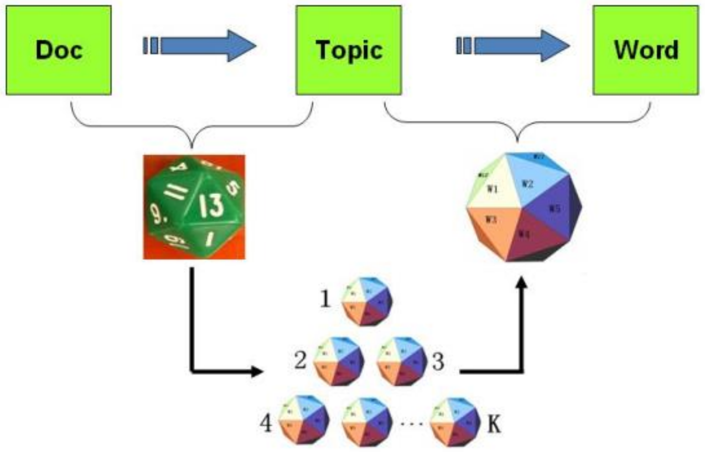
$$p(w|d_m) = \sum_{z=1}^{K} p(w|z)p(z|d_m) = \sum_{z=1}^{K} \varphi_{zw}\theta_{mz}$$

- 所以整篇文档的生成概率为

$$p(\vec{w}|d_m) = \prod_{i=1}^{n}\sum_{z=1}^{K} p(w_i|z)p(z|d_m) = \prod_{i=1}^{n}\sum_{z=1}^{K} \varphi_{zw_i}\theta_{dz}$$

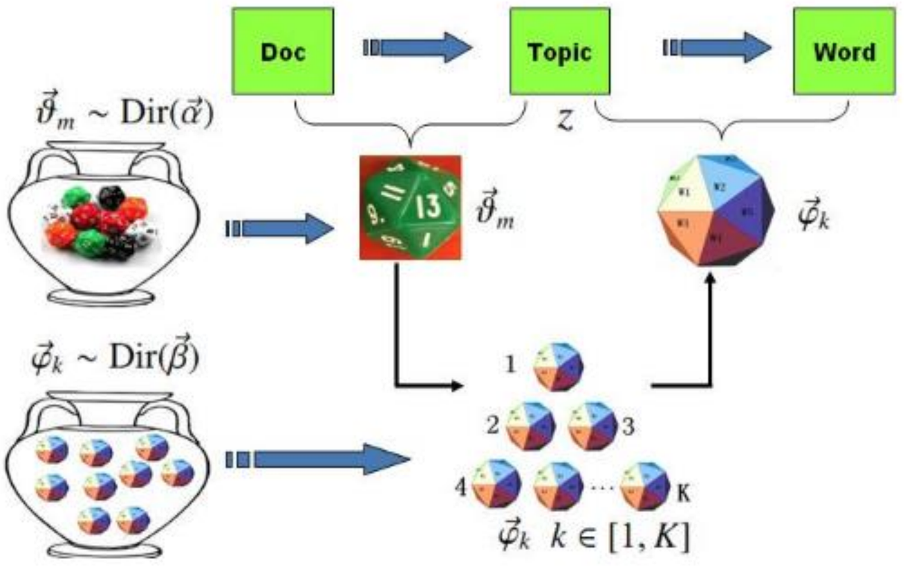


PLSA 模型的文档生成过程



Latent Dirichlet Allocation (LDA)

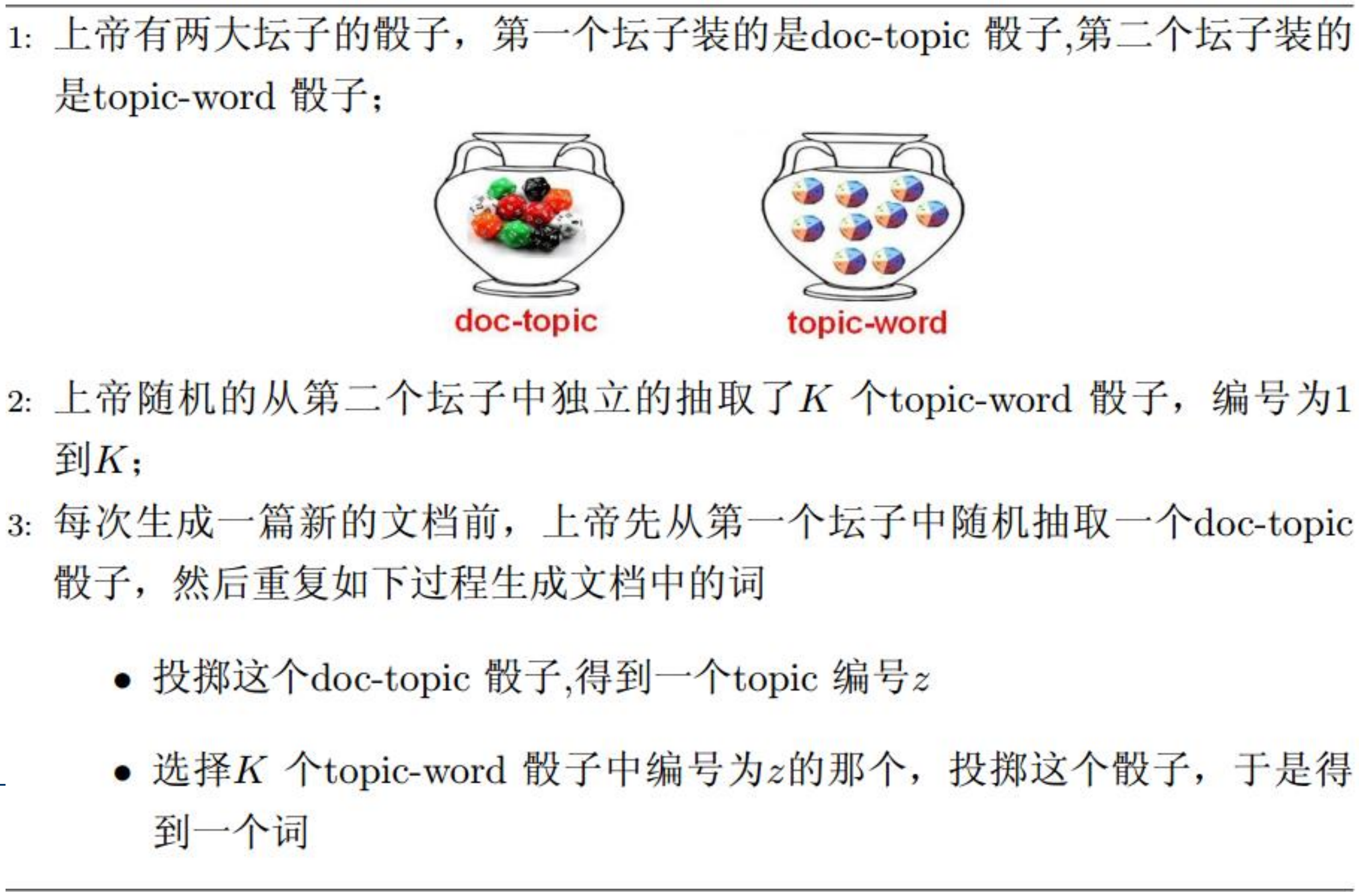
对于上述的 PLSA 模型，贝叶斯学派显然是有意见的，doc-topic 骰子 $\overrightarrow{\theta}_m$ 和 topic-word 骰子 $\overrightarrow{\varphi}_k$ 都是模型中的参数，参数都是随机变量，怎么能没有先验分布呢？于是，类似于对 Unigram Model 的贝叶斯改造， 我们也可以如下在两个骰子参数前加上先验分布从而把 PLSA 对应的游戏过程改造为一个贝叶斯的游戏过程。由于 $\overrightarrow{\varphi}_k$ 和 $\overrightarrow{\theta}_m$ 都对应到多项分布，所以先验分布的一个好的选择就是Drichlet 分布，于是我们就得到了 LDA(Latent Dirichlet Allocation)模型。



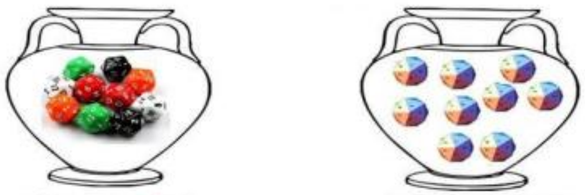
Doc
Topic
Word
z
$\vec{\vartheta}_m \sim \mathrm{Dir}(\vec{\alpha})$
$\vec{\vartheta}_m$
$\vec{\varphi}_k$
$\vec{\varphi}_k \sim \mathrm{Dir}(\vec{\beta})$
1
2
3
4
K
$\vec{\varphi}_k \quad k \in [1, K]$

Latent Dirichlet Allocation (LDA)

- **在 LDA 模型中, 上帝是按照如下的规则生成文档中单词的：**



1: 上帝有两大坛子的骰子，第一个坛子装的是doc-topic 骰子,第二个坛子装的是topic-word 骰子；



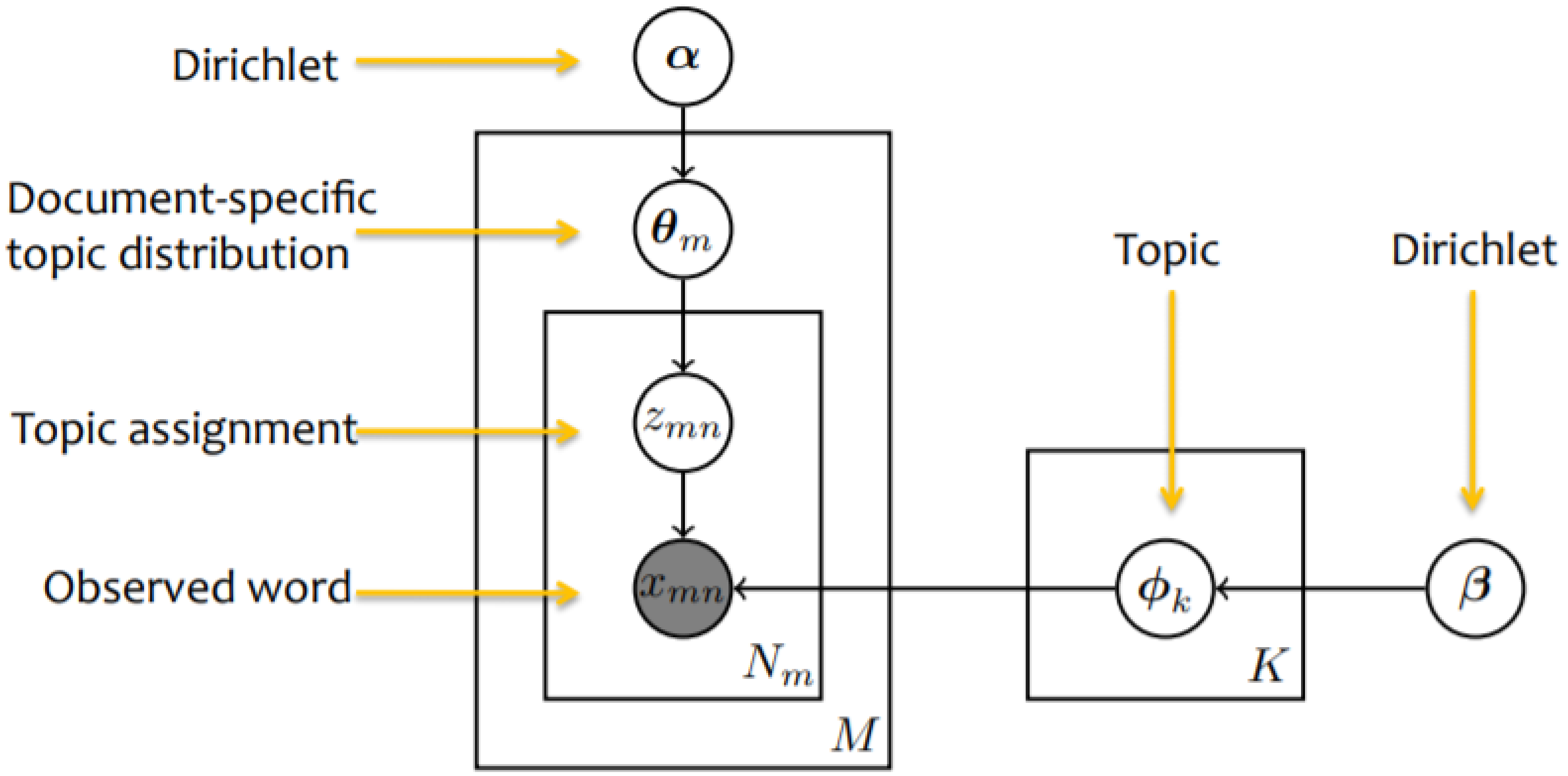
2: 上帝随机的从第二个坛子中独立的抽取了K 个topic-word 骰子，编号为1到K；

3: 每次生成一篇新的文档前，上帝先从第一个坛子中随机抽取一个doc-topic 骰子，然后重复如下过程生成文档中的词

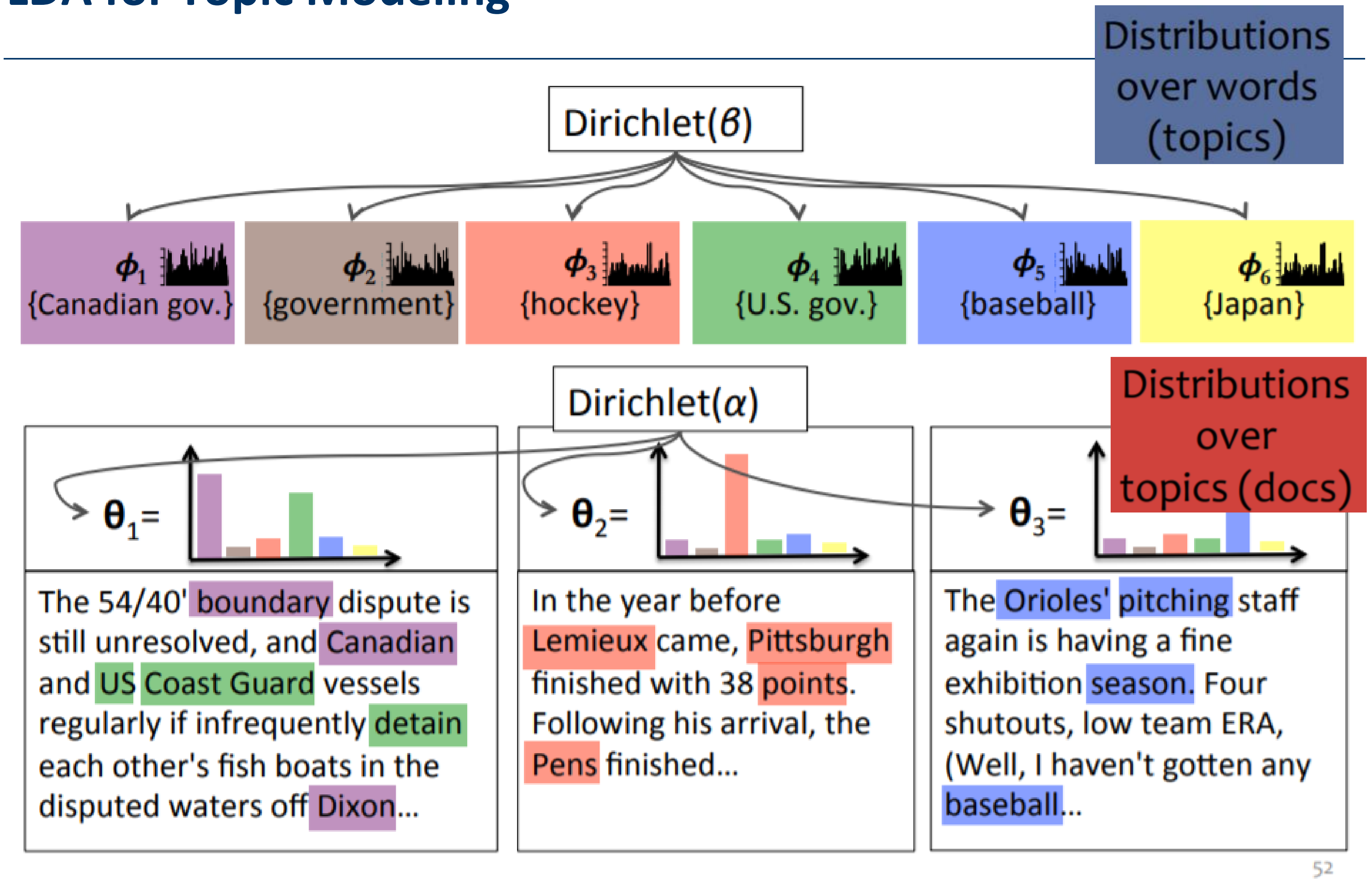
- 投掷这个doc-topic 骰子,得到一个topic 编号z
- 选择K 个topic-word 骰子中编号为z的那个，投掷这个骰子，于是得到一个词

Latent Dirichlet Allocation (LDA)

- **Plate Diagram**



LDA for Topic Modeling



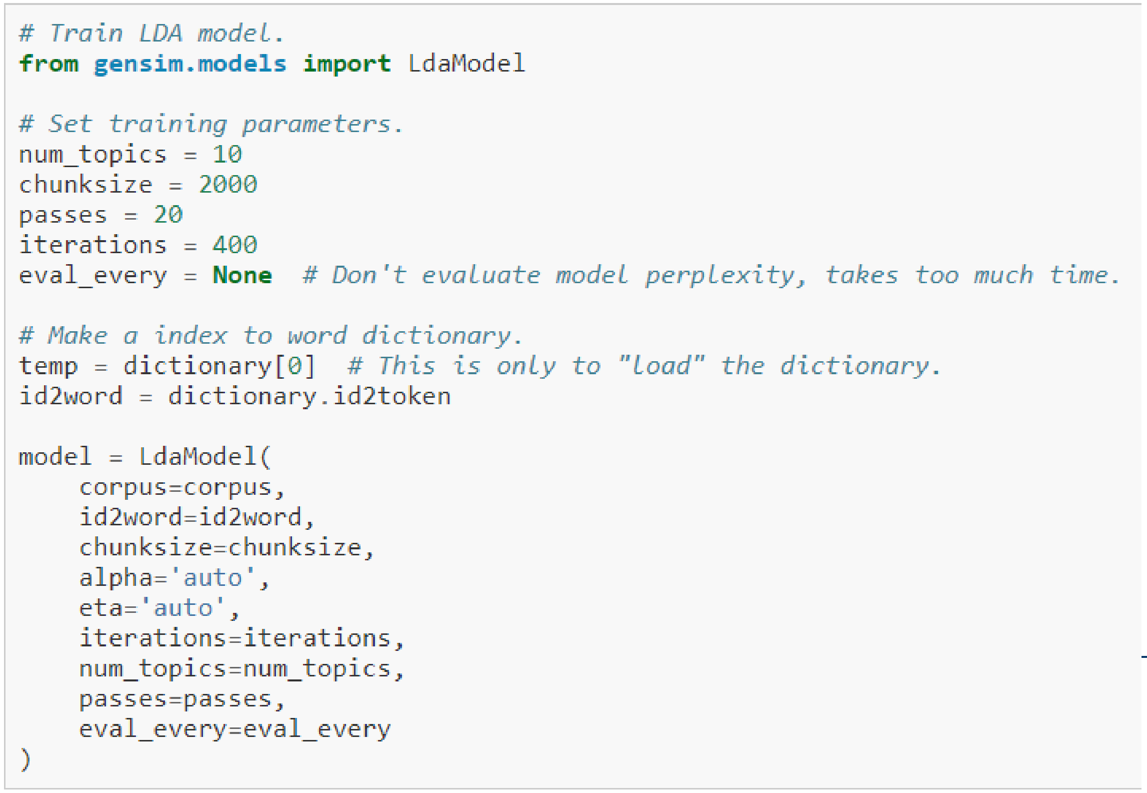
The 54/40' boundary dispute is still unresolved, and Canadian and US Coast Guard vessels regularly if infrequently detain each other's fish boats in the disputed waters off Dixon...

In the year before Lemieux came, Pittsburgh finished with 38 points. Following his arrival, the Pens finished...

The Orioles' pitching staff again is having a fine exhibition season. Four shutouts, low team ERA, (Well, I haven't gotten any baseball...

How to Use Topic Models?

- **A tool package: GENSIM**
  - https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html#sphx-glr-auto-examples-tutorials-run-lda-py
  - https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
  - Train LDA model



```python
# Train LDA model.
from gensim.models import LdaModel

# Set training parameters.
num_topics = 10
chunksize = 2000
passes = 20
iterations = 400
eval_every = None  # Don't evaluate model perplexity, takes too much time.

# Make a index to word dictionary.
temp = dictionary[0]  # This is only to "load" the dictionary.
id2word = dictionary.id2token

model = LdaModel(
    corpus=corpus,
    id2word=id2word,
    chunksize=chunksize,
    alpha='auto',
    eta='auto',
    iterations=iterations,
    num_topics=num_topics,
    passes=passes,
    eval_every=eval_every
)
```

How to Use Topic Models?

- **For recommendation,**
  - The key is to acquire a topic distribution of each document.
  - The distribution is the low-dimensional representation of a document
  - The you can perform textual similarity computation.

Distributional Representation

Problems with this discrete representation

- In vector space terms, this is a vector with one 1 and a lot of zeroes

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$$

  - Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

- We call this a “one-hot” representation. Its problem:

$$\begin{array}{ll} \text{motel} & [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \quad \text{AND} \\ \text{hotel} & [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \quad = \ 0 \end{array}$$

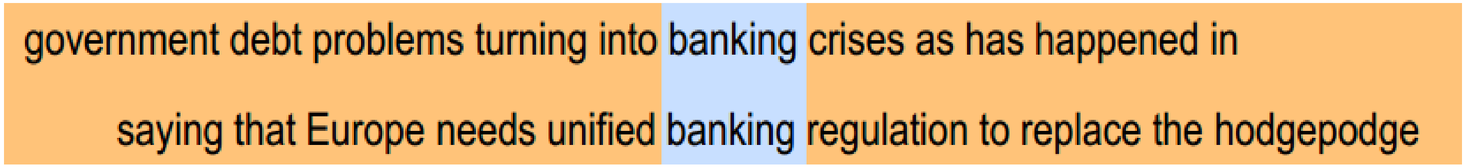
Distributional similarity based representations

- You can get a lot of value by represen4ng a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical NLP



government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

How to make neighbors represent words?

- Answer: With a cooccurrence matrix X
  - 2 options: full document v.s. windows
  - Word - document cooccurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
  - Instead: Window around each word → captures both syntactic (POS) and semantic information

Window based cooccurence matrix

- **Window length 1 (more common: 5-10)**
- **Example corpus:**
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with simple cooccurrence vectors

- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues

Word2Vec

- Instead of capturing cooccurrence counts directly,
- Predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

Details of Word2Vec

- Predict surrounding words in a window of length m of every word.
- Objective function: Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$

  - Where θ represents all variables we optimize

Details of Word2Vec

- Predict surrounding words in a window of length m of every word
- For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

  - where o is the outside (or output) word id, c is the center word id, u and v are “center” and “outside” vectors of o and c
- Every word has two vectors
- This is essentially “dynamic” logistic regression

Linear Relationships in word2vec

- These representations are *very good* at encoding dimensions of similarity!
  - Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

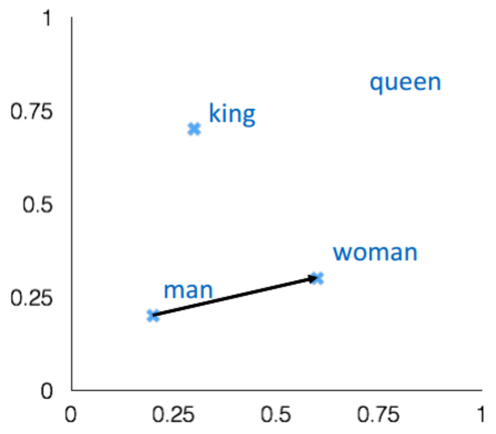
- $X_{apple} - X_{apples} \approx X_{car} - X_{cars} \approx X_{family} - X_{families}$

Test for linear relationships, examined by Mikolov et al. (2014)

$$\boxed{a:b :: c:?} \longrightarrow \boxed{d = \arg\max_{x} \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}}$$

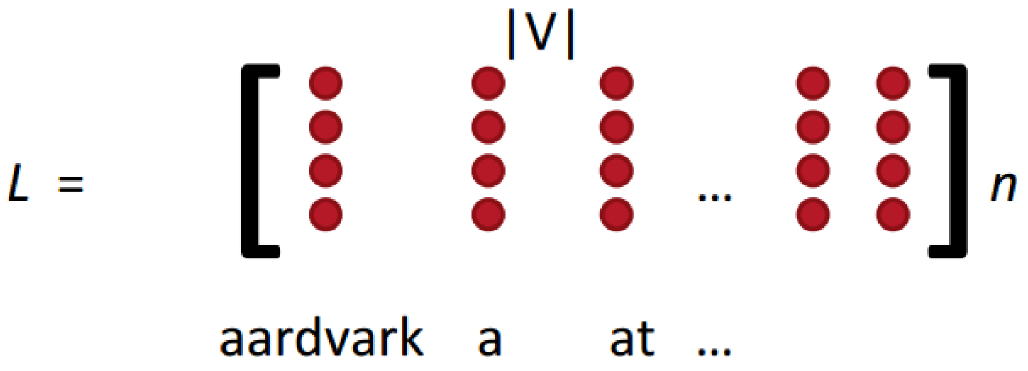
man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Word Embedding Matrix

- Initialize most word vectors of future models with our "pretrained" embedding matrix $L \in \mathbb{R}^{n\times|V|}$

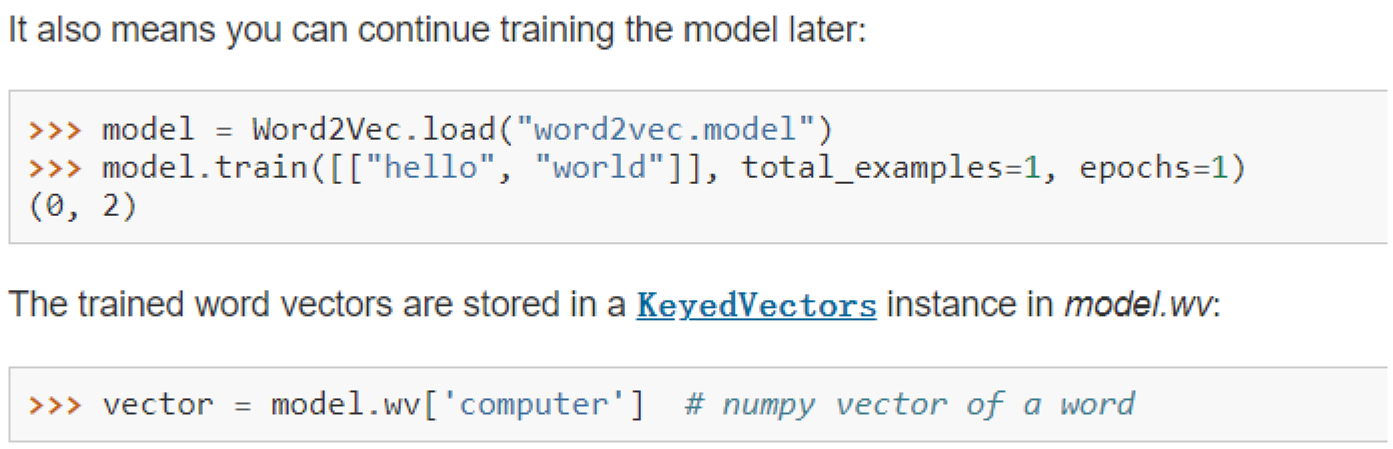


- Also called a look-up table
  - Conceptually you get a word's vector by left multiplying a one-hot vector *e* (of length |V|) by *L*: *x* = *Le*

How to use low dimensional word vectors?

- **Document embedding:**
  - A simple mean summation of word embeddings

- **Call API:**
  - GENSIM:
    - https://radimrehurek.com/gensim/models/word2vec.html



It also means you can continue training the model later:

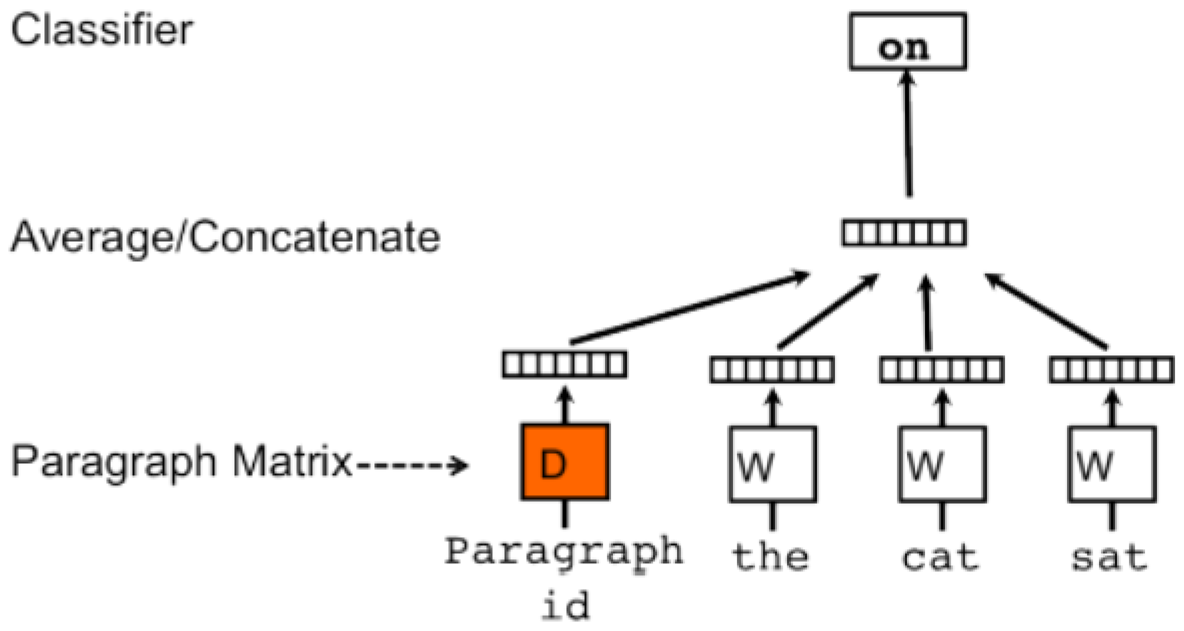
```
>>> model = Word2Vec.load("word2vec.model")
>>> model.train([["hello", "world"]], total_examples=1, epochs=1)
(0, 2)
```

The trained word vectors are stored in a KeyedVectors instance in *model.wv*:

```
>>> vector = model.wv['computer']  # numpy vector of a word
```

Doc2Vec

- **Architecture**
  - Introducing document embedding into optimization.
  - GENSIM:
    - https://radimrehurek.com/gensim/models/doc2vec.html


Classifier
on
Average/Concatenate
Paragraph Matrix
D
W
W
W
Paragraph id
the
cat
sat