Unsupervised Learning for Text

Joy Rimchala
IDEA DS&A
Intro to Data Science Series
2015.11.05

Motivation

- AI can help process human language to “understand” and “communicate” with human — elusive, long term

- Unsupervised NLP as a better tools — short term, more pragmatic
  - improve text search
  - information extraction, topic finding/discovery
  - sentiment analysis for marketing or campaign
  - automated/assisted machine translation
  - complex question answering

What's covered in this talk . . .

- Features from text (counts & frequencies) ~ 5 mins
  - word count (bag-of-word) and TF-IDF
  - word co-occurrence matrix
- Topic modeling ~ 15-20 mins
  - Latent Dirichlet Allocation (LDA)
- Vector Representation of Text ~ 10-15 mins
  - word2vec

Simple Feature Generation:
Bag of Word representation (BOW)

- Word counts - bag of word representation
  - Example codes in Notebook [1]-[3]
  - for each document,
    - split words by delimiter (usually white space)[1]
    - (additional pre-processing: removing stop words, stemming, lemmatizing)[1]
    - count word frequencies[3]

BOW in Python version I

```
file_name = './ted_mini/art_positive/5.ted'
delim = " "
with open(file_name, 'r') as f:
    dat = f.readlines()
    dat = map(lambda x: x.replace("\n", "").lower().split(delim), dat)
```

```
"""what i want to talk
this is actually quite
and so cars , as art ,
now at this point you '
```

```
['what', 'want', 'to', 'talk'
s', 'actually', 'quite', 'mea
'on', 'the', 'totem', 'pole',
f', 'it', 'and', 'cars', 'are
t', 'into', 'the', 'aesthetic
```

	word	count
0	young	639
1	york	638
2	yelled	637
3	year	636
4	wrist	635

```
d = pd.DataFrame(list(chain(*dat))).rename(columns={0:'word'})
d['count'] = 1
df = d.groupby('word').agg('sum').sort('count', ascending=False)
```

BOW in Python version II

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
vec = cv.fit(dat)
```

```
"""what i want to talk
this is actually quite
and so cars , as art ,
now at this point you '
```

```
['what', 'want', 'to', 'talk'
s', 'actually', 'quite', 'mea
'on', 'the', 'totem', 'pole',
f', 'it', 'and', 'cars', 'are
t', 'into', 'the', 'aesthetic
```

	word	count
0	young	639
1	york	638
2	yelled	637
3	year	636
4	wrist	635

```
df = pd.DataFrame(
    vec.vocabulary_.items())
        .rename(columns={0:'word', 1:'count'})
        .sort('count', ascending=0)
)
```

Simple Feature Generation:

Term Freq-Inverse Document Freq (TF-IDF)

- Term Frequency (TF): Normalized word counts:
  - Per document count of #times word appears in a document normalized by total number of words in document (i.e. per document BOW)
- Document Frequency (DF)
  - #documents containing a token normalized by total number of documents
- Inverse Document Frequency (IDF)
  - 1/DF

Computing TF-IDF

- Corpus with 3 documents:

d_1: I like data science and data discovery. (7)

d_2: I think data science requires data exploration and machine learning (10)

d_3: I apply machine learning to data for science discovery (9)

- Corpus:

{I, like, data, science, and, discovery, think, require, exploration, machine, learning, apply, to, for}

- Term Frequency (TF):

d_1: {I: 1/7, like: 1/7, data: 2/7…}

d_2: {I: 1/10, think: 1/10, data: 2/10, …}

d_3: {I: 1/9, apply: 1/9, machine: 1/9, …}

- Document Frequency (DF):

{I: 3/3, like: 1/3, data: 3/3, science: 3/3, and: 2/3, discovery: 2/3 , …}

TF-IDF in Python

Poorman's version

```
def tf(t, doc):
    if type(doc) == list:
        return float(doc.count(t))/float(len(doc))
    elif type(doc) == str:
        doc = text_to_bagofwords(doc)
        return float(doc.count(t))/float(len(doc))
    else:
        return NaN

def idf(t, corpus):
    df = 0
    for doc in corpus:
        if doc.count(t) > 0:
            df += 1
    return float(len(corpus))/float(df + 0.01)

def tfidf(t, doc, corpus):
    return tf(t, doc)*idf(t, corpus)
```

Off the shelf package

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
tfidf = vec.fit_transform(dat)
```

What's topic modeling ?

- Topic modeling
  - discovers main themes that pervade collection of documents.
  - organizes the collection according to the discovered themes.
  - is unsupervised.
- Topic modeling algorithms can be adapted to many kinds of data including:
  - genetic data
  - images
  - social networks

Why topic modeling ?

- Get major themes or topics in text
  - Huge amount of documents
  - Want to know what's going on but can't read them all
- Unsupervised
- Simple way to analyze unlabeled text

Quick Review I

Multinomial distribution

- Given an observed sequence of die rolling what's the probability of an observed set of die outcome
- Die rolling is "generated" by a "hidden (Multinomial) process"
- The probability of this outcome set is described by <u>Multinomial distribution</u>

For topic modeling

- Three step dice rolling
  - per document topic distribution
  - topic die given topic distribution
  - word choice die given topic

Quick Review II

Chain Rule and Bayes' Rule

Chain Rule: $$Pr(A\ and\ B) = Pr(A|B) \cdot Pr(A)$$

Bayes' Rule:

$$\underbrace{Pr(B|A)}_{\text{posterior}} = \frac{\overbrace{Pr(A|B)}^{\text{likelihood}} \cdot \overbrace{Pr(A)}^{\text{prior}}}{\underbrace{Pr(B)}_{\text{evidence}}}$$

For die rolling

- A = observed data (set of observed outcome)
- B = model parameters (probability of getting each face)

For topic modeling

- A = observed data (set of observed topic outcome)
- B = model parameters (probability of getting each topic per document, of getting word per topic, of getting topic proportion per document)

Quick Review III

Posterior Inference & Conjugate Prior

- Posterior Inference:
  - inferring model parameters from observed likelihood and prior
  - Likelihood: $Pr(A|B)$
    - Prob. of observing set of dice rolling outcome A given that it is generated by multinomial process with parameter set B
  - Prior: $Pr(A)$
    - Underlying knowledge about model parameters
    - if not know used “unbiased prior”

- Conjugate Prior:
  - posterior distribution is from the same family as prior time posterior
  - make the math for posterior inference easier
  - Conjugate prior of Multinomial distribution is **Dirichlet distribution**

Topic modeling algorithm:
Latent Dirichlet Allocation (LDA)

- Invented by David Blei (Toronto), Andrew Ng (Stanford), and Michael I. Jordan (Berkeley) as improvement to LSI and LSA
- Assume documents exhibit multiple topics
  - document ~ multinomial distribution over topics
  - Dirichlet distribution is a conjugate prior of multinomial

- Hidden topics in documents are generated by a 3-level Multinomial process
  - process is described by topic weights
  - topic is a distribution over words
  - words have different level of “membership” to a topic
  - documents consists of multiple topics in different proportion

LDA as three step nested Multinomial Process

$$\underbrace{Pr(B|A)}_{\textbf{posterior}} = \frac{\overbrace{Pr(A|B)}^{\textbf{likelihood}} \cdot \overbrace{Pr(A)}^{\textbf{prior}}}{\underbrace{Pr(B)}_{\textbf{evidence}}}$$

Word choosing die

- A = observed word frequencies in document
- B = model parameters (probability of getting each number for fair die)

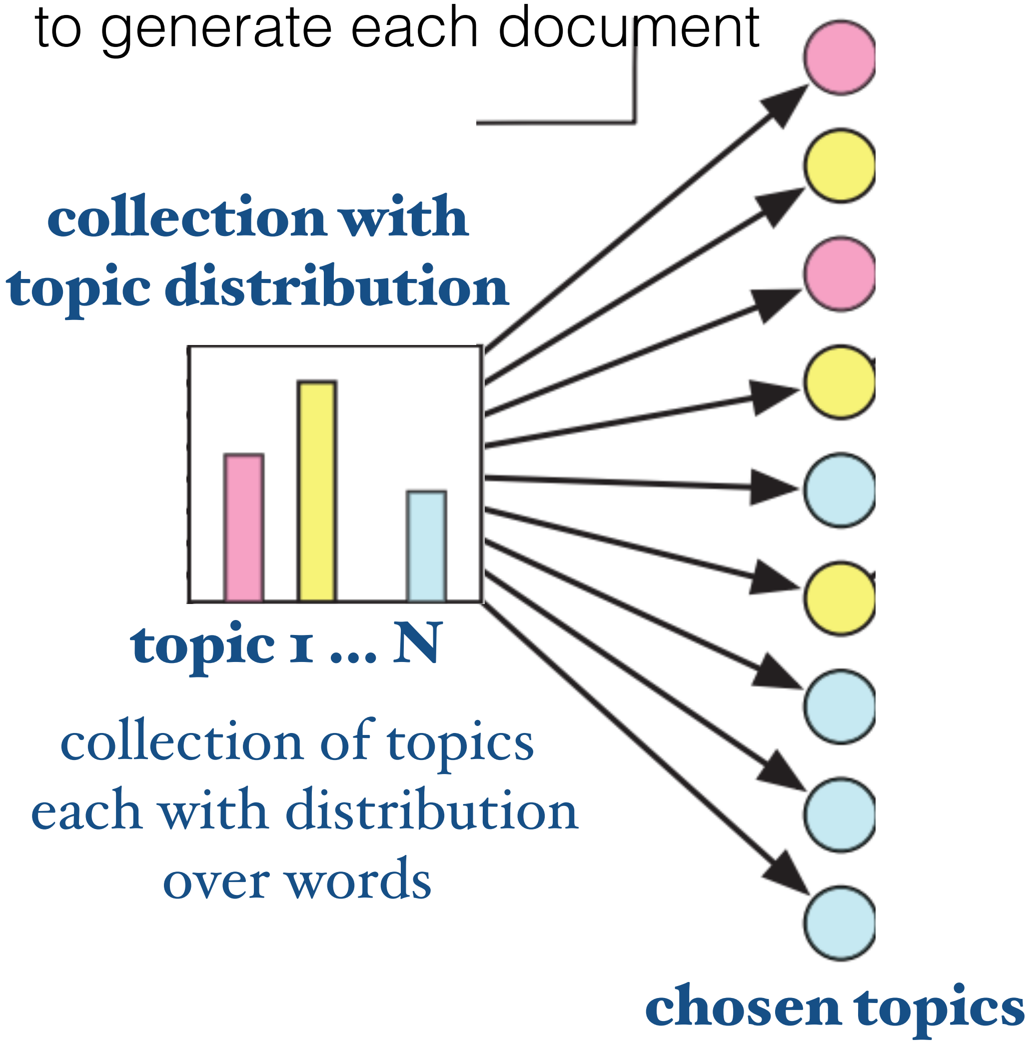
Topic choosing die

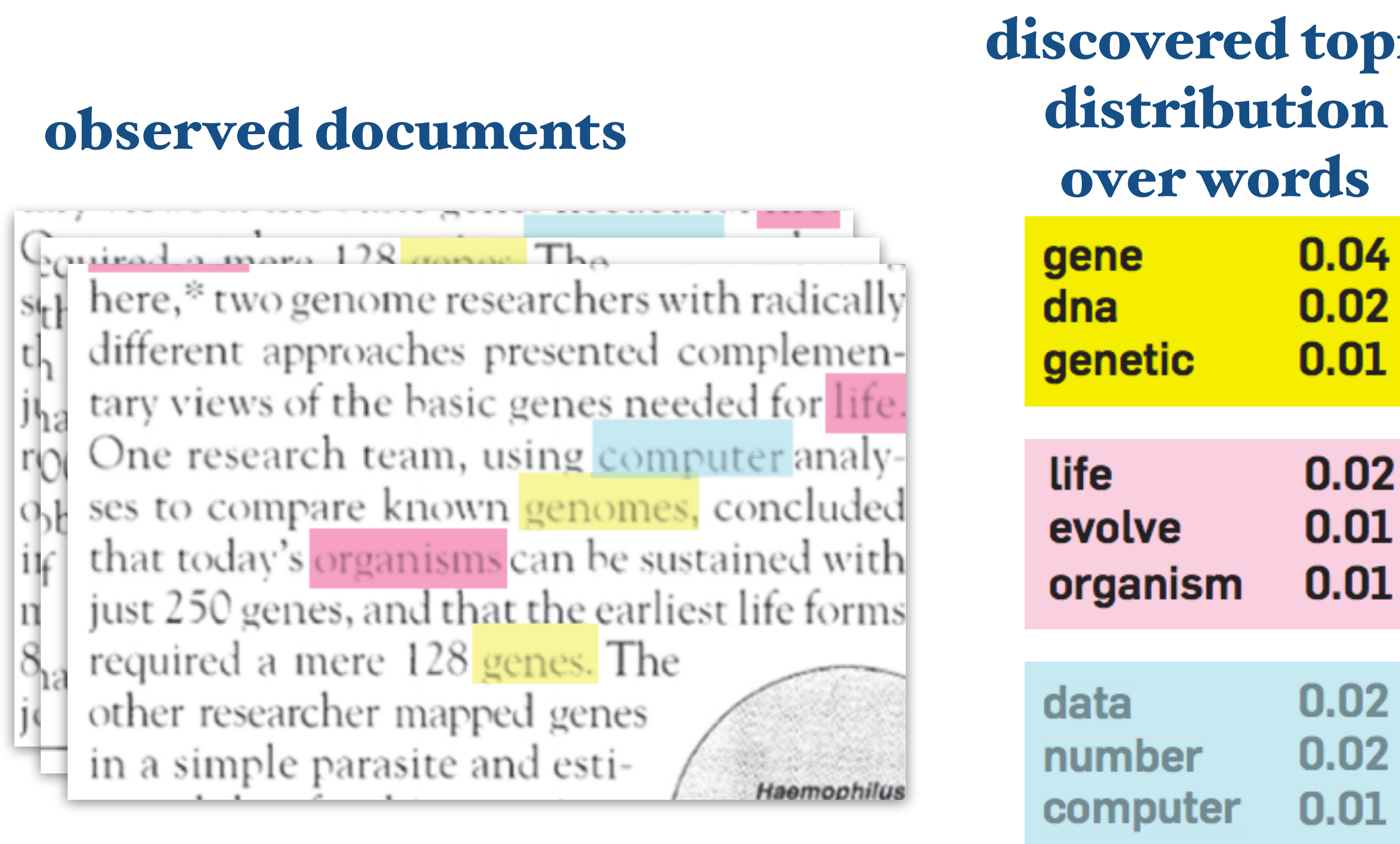
- A = observed topic distributions in document
- B = model parameters (probability of getting each topic given per topic distribution)

LDA assumes documents are generated by (hidden) Dirichlet Process

1. choose one of the distributions over words to generate each document

3. Repeat for all documents in collections



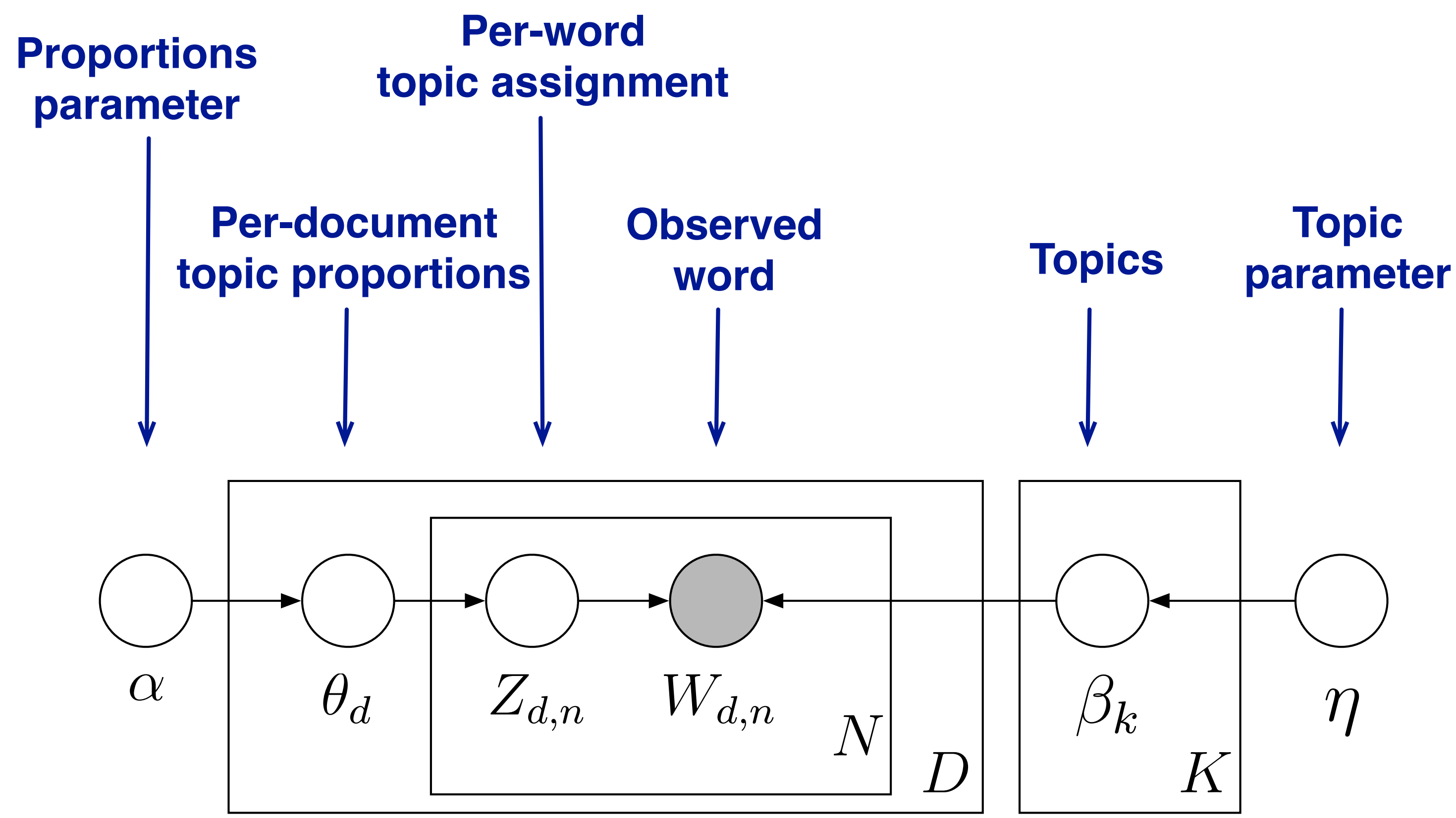


- In reality, observed documents consisting of words
- LDA takes all documents and infers underlying hidden distribution (structure) of topics

2. Choose word from one of the topics (blue, yellow, pink) and look up probability of chosen word in that topic

Image Taken from: http://www.cs.princeton.edu/~blei/papers/Blei2012.pdf

LDA as graphical model



what kind of die generate the document, holding topic fix

LDA: reverse the generative process

Draw each topic for all topics 1, … ,K β_k

For each document:

Draw topic proportion θ_d

For each word

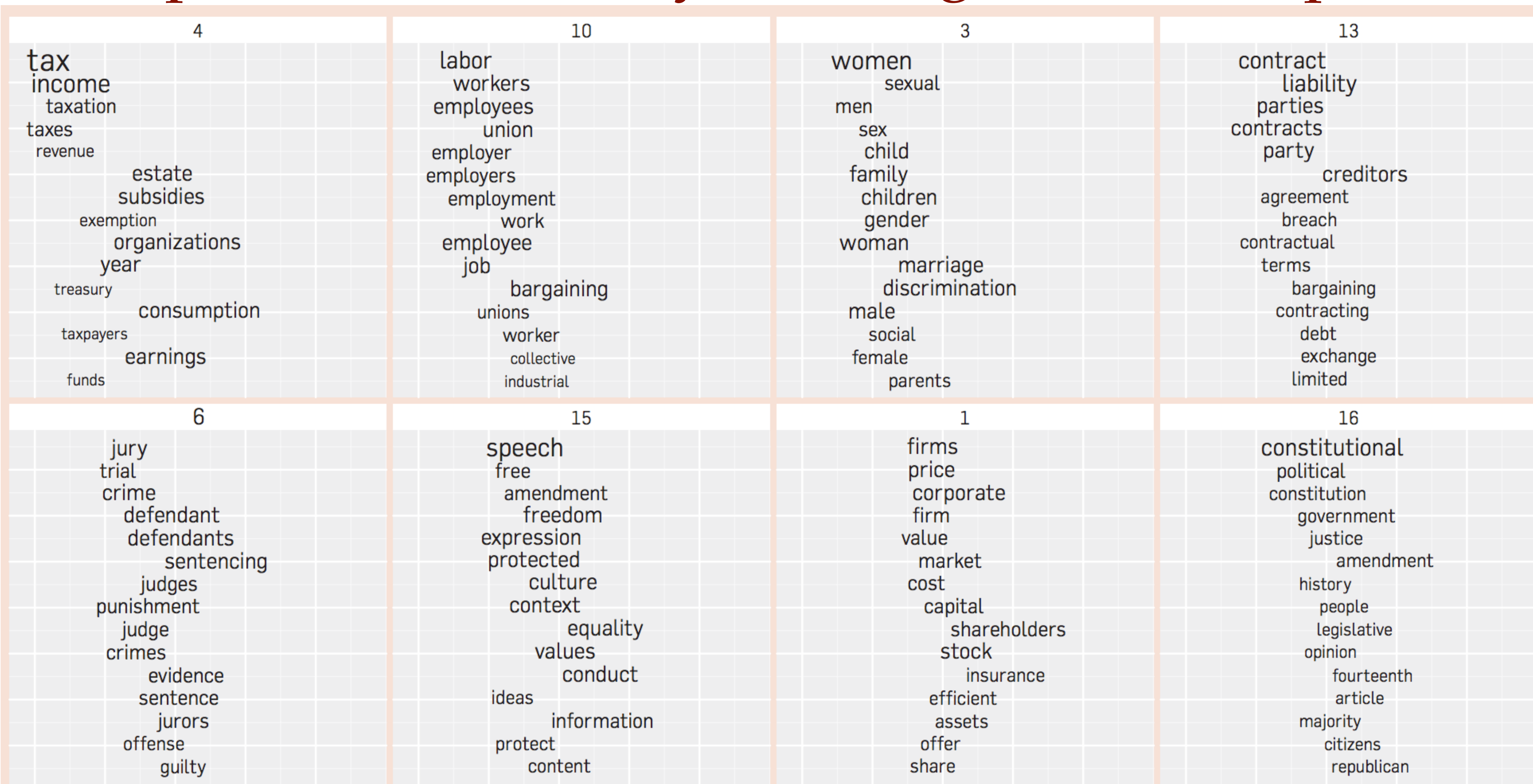
Draw topic assignment $Z_{d,n}$

Draw word $W_{d,n}$

Image Taken from:
http://www.cs.princeton.edu/~blei/papers/Blei2012.pdf

Example LDA Results:

Topic auto-discovery from legal cases corpus



topic is a distribution over words
words have different level of “membership” to a topic
documents consists of multiple topics in different proportion

ResultsTaken from:
http://www.cs.princeton.edu/~blei/papers/Blei2012.pdf

LDA Input Computation Co-occurrence Matrix by Hand

d_1: I like data science and data discovery. (7)

d_2: I think data science requires data exploration and machine learning (10)

d_3: I apply machine learning to data for science discovery (9)

	I	like	data	scien ce	and	disco very	think	requi re	explo ratio	mach ine	learni ng	apply	to	for
I	0	1	3	3	2	2	1	1	1	1	1	1	1	1
like	1	0	1	1	1	1	0	0	0	0	0	0	0	0
data	3	1	0	3	2	2	1	1	1	2	2	1	1	1
science	3	1	3	0	2	1	1	1	1	2	2	1	1	1
and	2	1	2	2	0	1	1	1	1	1	1	0	0	0
discovery	2	1	2	1	1	0	1	0	0	1	1	1	1	1
think	1	0	1	1	1	1	0	1	1	1	1	0	0	0
require	1	0	1	1	1	0	1	0	1	1	1	0	0	0
exploration	1	0	1	1	1	0	1	1	0	1	1	0	0	0
machine	1	0	2	2	1	1	1	1	1	0	3	1	1	1
learning	1	0	2	2	1	1	1	1	1	3	0	1	1	1
apply	1	0	1	1	0	1	0	0	0	1	1	0	1	1
to	1	0	1	1	0	1	0	0	0	1	1	1	0	1
for	1	0	1	1	0	1	0	0	0	1	1	1	1	0

LDA in Python lda

Reuters News corpus

```
import lda
model = lda.LDA(n_topics=20, n_iter=1500, random_state=1)
model.fit(X)
ntop=10
for i, topic_dist in enumerate(model.topic_word_[:ntop]):
    topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(ntop+1):-1]
    print('Topic {}: {}'.format(i, ' '.join(topic_words)))
```

```
Topic 0: british churchill sale million major letters west
Topic 1: church government political country state people
Topic 2: elvis king fans presley life concert young death
Topic 3: yeltsin russian russia president kremlin moscow m
Topic 4: pope vatican paul john surgery hospital pontiff r
Topic 5: family funeral police miami versace cunanan city
Topic 6: simpson former years court president wife south c
Topic 7: order mother successor election nuns church nirma
Topic 8: charles prince diana royal king queen parker bowl
Topic 9: film french france against bardot paris poster an
```

Example taken from https://ariddell.org/lda.html

LDA in Python gensim

TED corpus

```python
from gensim.models.ldamodel import LdaModel
lda = LdaModel(corpus=artsci_corpus, id2word=dictionary,
               num_topics=100, update_every=1, chunksize=100, passes=1)
```

```python
## Example topic
lda.show_topic(11)
```

```
[(0.015640414022064685, 'earth'),
 (0.012160429080861469, 'stars'),
 (0.011924733515645118, 'like'),
 (0.011199539194868741, 'universe'),
 (0.010963916617258741, 'atoms'),
 (0.0095423565216509812, 'century'),
 (0.0085391585978015876, 'einstein'),
 (0.0079199531795733497, 'billion'),
 (0.0074580017159042661, 'planets'),
 (0.0073421488411868473, 'small')]
```

```python
## Example topic
lda.show_topic(15)
```

```
[(0.019069935795224396, 'ants'),
 (0.012175520975191553, 'nest'),
 (0.0096068740689464944, 'ant'),
 (0.0095467406982767469, 'people'),
 (0.0092058530459285474, 'colony'),
 (0.0072401546937930416, 'like'),
 (0.0069742669948207143, 'workers'),
 (0.0062076249103784975, 'years'),
 (0.0060125350934175431, 'city'),
 (0.0054315333587530582, 'world')]
```

More information and example codes at:
https://radimrehurek.com/gensim/models/ldamodel.html

An Efficient discrete representation

Word2Vec

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

- Invented by Tomas Mikolov implemented in C with Python binding by Radim Řehůřek
- Assume that words that occur together share some semantic relationship
- How to make neighbors represent word ?
    - Window based word-word co-occurrence matrix
    - Predict surrounding words of every word in a window of length c

government debt problems turning into banking crises as has happened in

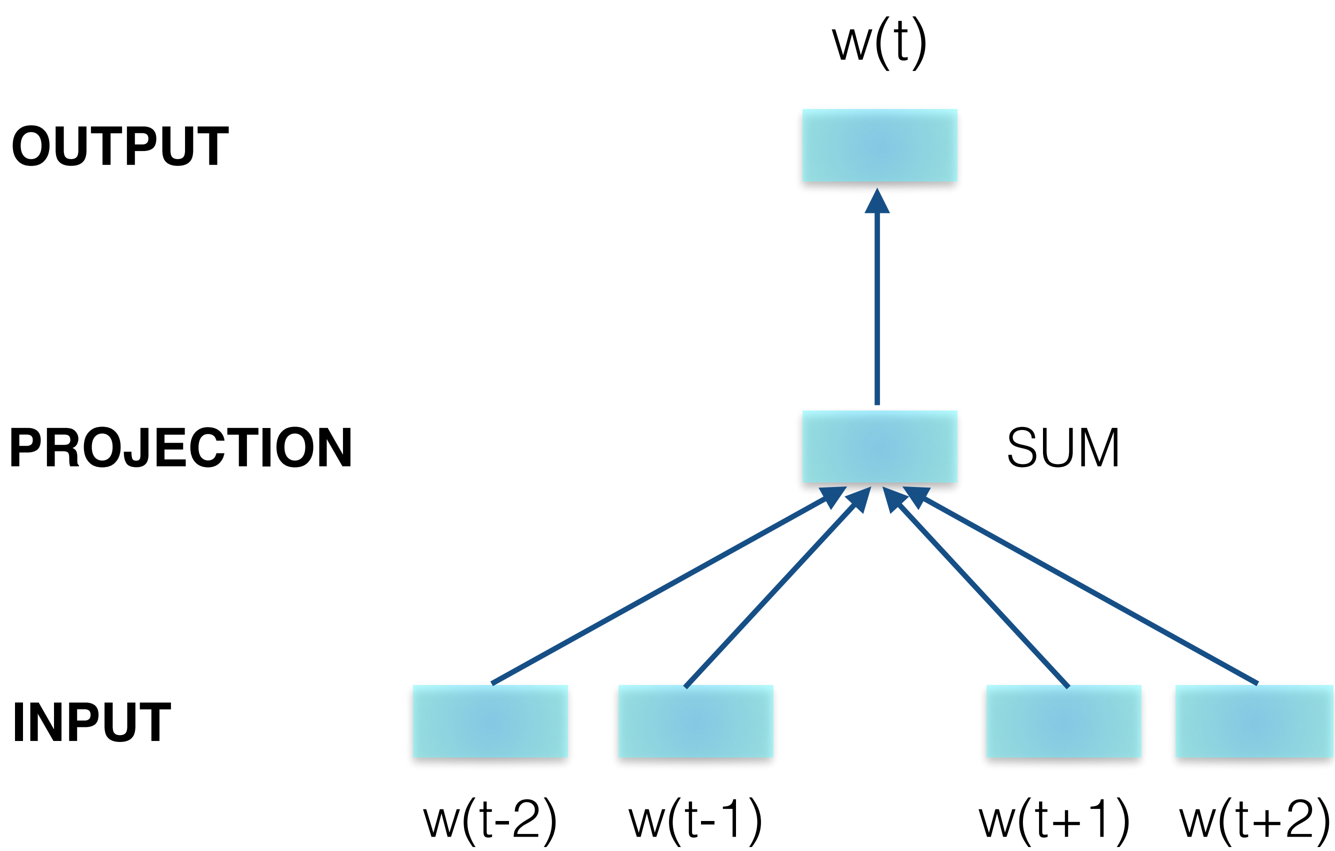
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

From Words to Vectors:

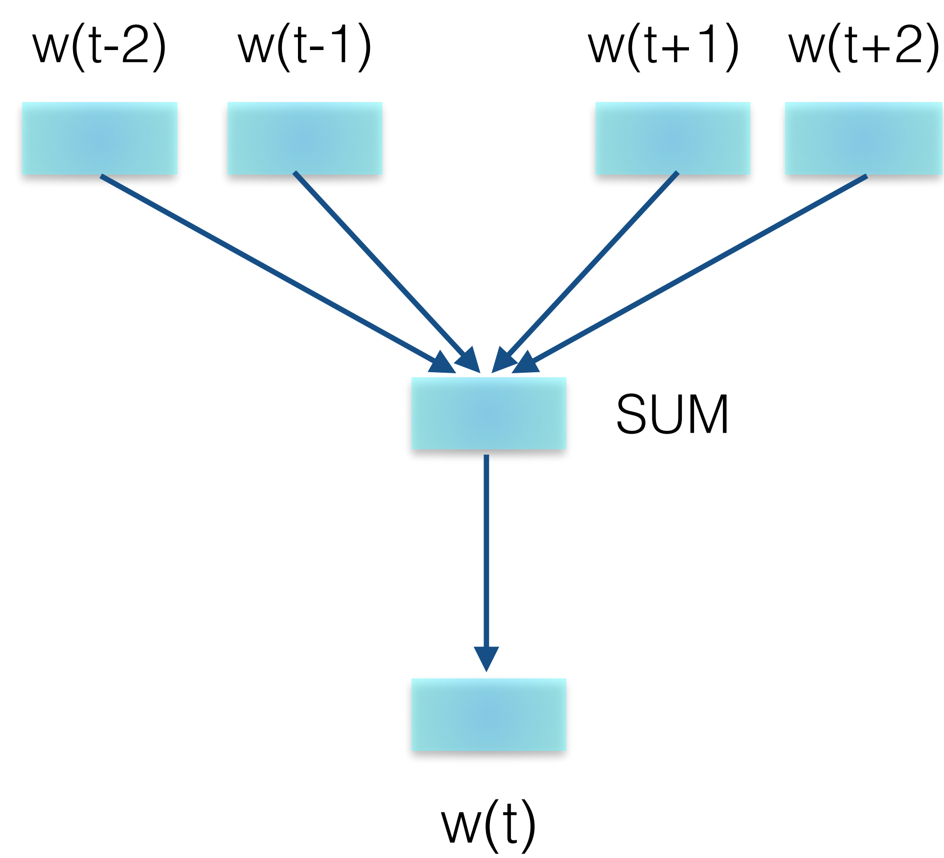
the two well known architectures

continuous bag of words (CBOW)



CBOW predicts the current word (inner vector) based on the surrounding words (outer vector, context)

Skip-Gram



Skip-Gram predicts surrounding words based on center word

From Words to Vectors:

Word2Vec Optimization problem

- Objective:
  - Maximize likelihood of any context word given current center word
  - Do this for every word in the corpus

Skip-Gram objective function:

maximize log-likelihood of center word given surrounding words within a specified window

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

Skip-Gram Likelihood function

softmax of dot products between center word (inner vector) and surrounding words (outer vectors)

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^\top v_{w_I}\right)}$$

Emerging Semantic Relationship in Word2Vec

Example Results

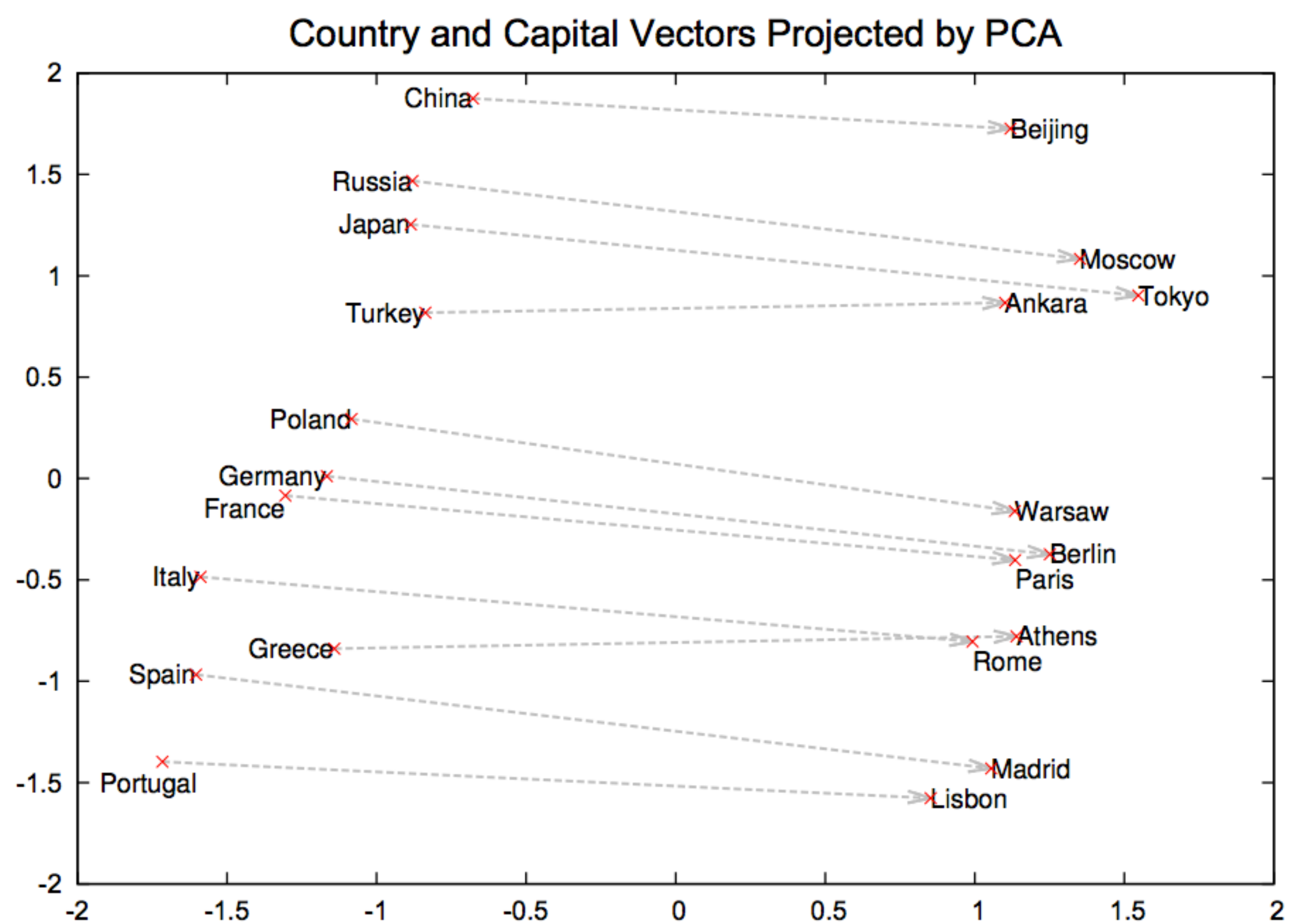
$X_{king} - X_{man} \sim X_{queen} - X_{woman}$

$X_{apple} - X_{apples} \sim X_{car} - X_{cars}$

$X_{dog} - X_{dogs} \sim X_{family} - X_{families}$

$X_{shirt} - X_{clothing} \sim X_{chair} - X_{furniture}$

Benchmark: Mikolov *et al* NIPS 2012



Skip-Gram Word2Vec in Python

```
from gensim.models.word2vec import Word2Vec
w2vmodel = Word2Vec(texts, size=100, window=5, min_count=5, workers=2)
w2vmodel.save(os.join.path(data_dir), 'artsci_positive_w2vmodel')
```

```
w2vmodel.similarity('man', 'woman')
```

```
0.7145058283370651l
```

```
model.most_similar(positive=['cambridge', 'brain'], negative=['pittsburgh'], topn=1)
```

```
[('protein', 0.6212430000305176)]
```

```
model.doesnt_match("breakfast food neurons dinner".split())
```

```
'neurons'
```

More information and example codes at:
https://radimrehurek.com/gensim/models/word2vec.html

Takeaways

- Simple features from text (counts & frequencies)
  - word count (bag-of-word) and TF-IDF: quick and easy to compute
  - word co-occurrence matrix: usually yield really sparse matrix
  - off-the-shelf in Python: **sklearn.feature_extraction.text** and **NLTK**

- Topic modeling: Latent Dirichlet Allocation (LDA)
  - Unsupervised method. Good for analyzing unlabeled text.
  - Computationally expensive to train, unclear measure of success (this is true for a lot of unsupervised learning algorithms)
  - off-the-shelf package in Python: **lda** and **gensim.model.lda** *(not sklearn.lda)*

- Vector Representation of Text (Skip-Gram word2vec)
  - Unsupervised method. Good for capturing semantic similarities across documents
  - off-the-shelf package in Python: **gensim.model.word2vec**