COMP7630 – Web Intelligence and its Applications

Web Information Retrieval
(part 1)

Valentino Santucci

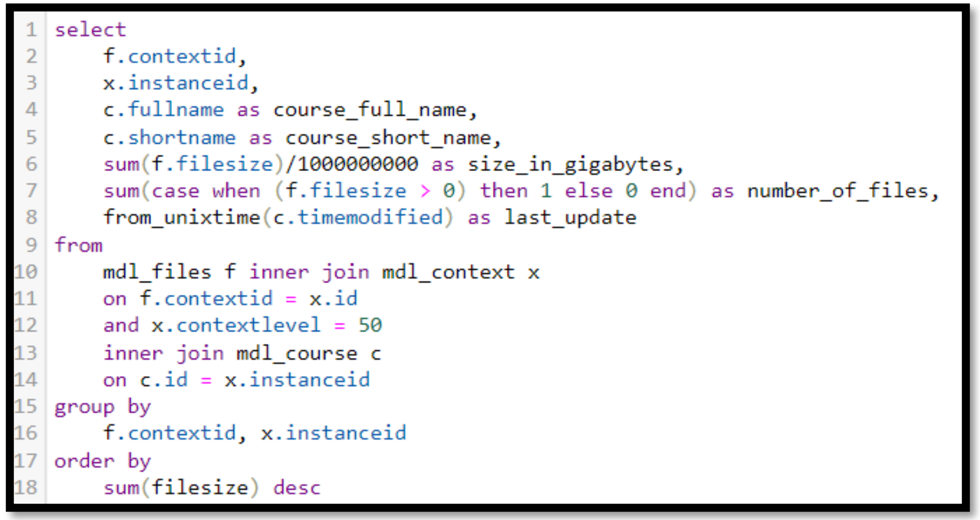
(valentino.santucci@unistrapg.it)

Outline

- <u>Goals and architecture of a IR system</u>
- IR models
- Word2Vec and other modern Language Models

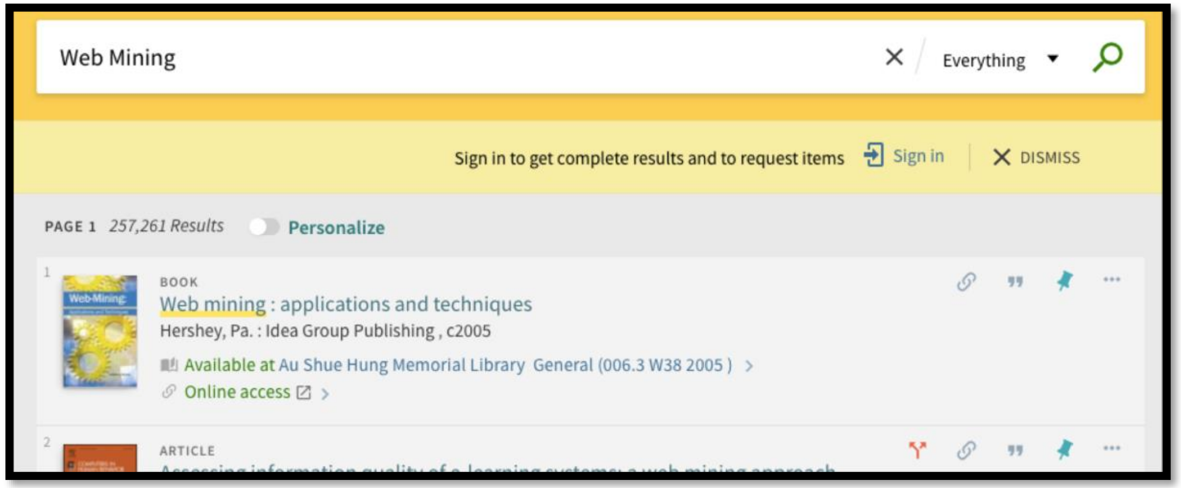
Information Retrieval vs Database Search

- Querying a relational database in a DBMS



```sql
select
    f.contextid,
    x.instanceid,
    c.fullname as course_full_name,
    c.shortname as course_short_name,
    sum(f.filesize)/1000000000 as size_in_gigabytes,
    sum(case when (f.filesize > 0) then 1 else 0 end) as number_of_files,
    from_unixtime(c.timemodified) as last_update
from
    mdl_files f inner join mdl_context x
    on f.contextid = x.id
    and x.contextlevel = 50
    inner join mdl_course c
    on c.id = x.instanceid
group by
    f.contextid, x.instanceid
order by
    sum(filesize) desc
```

- Querying an IR system



Sign in to get complete results and to request items

PAGE 1 257,261 Results

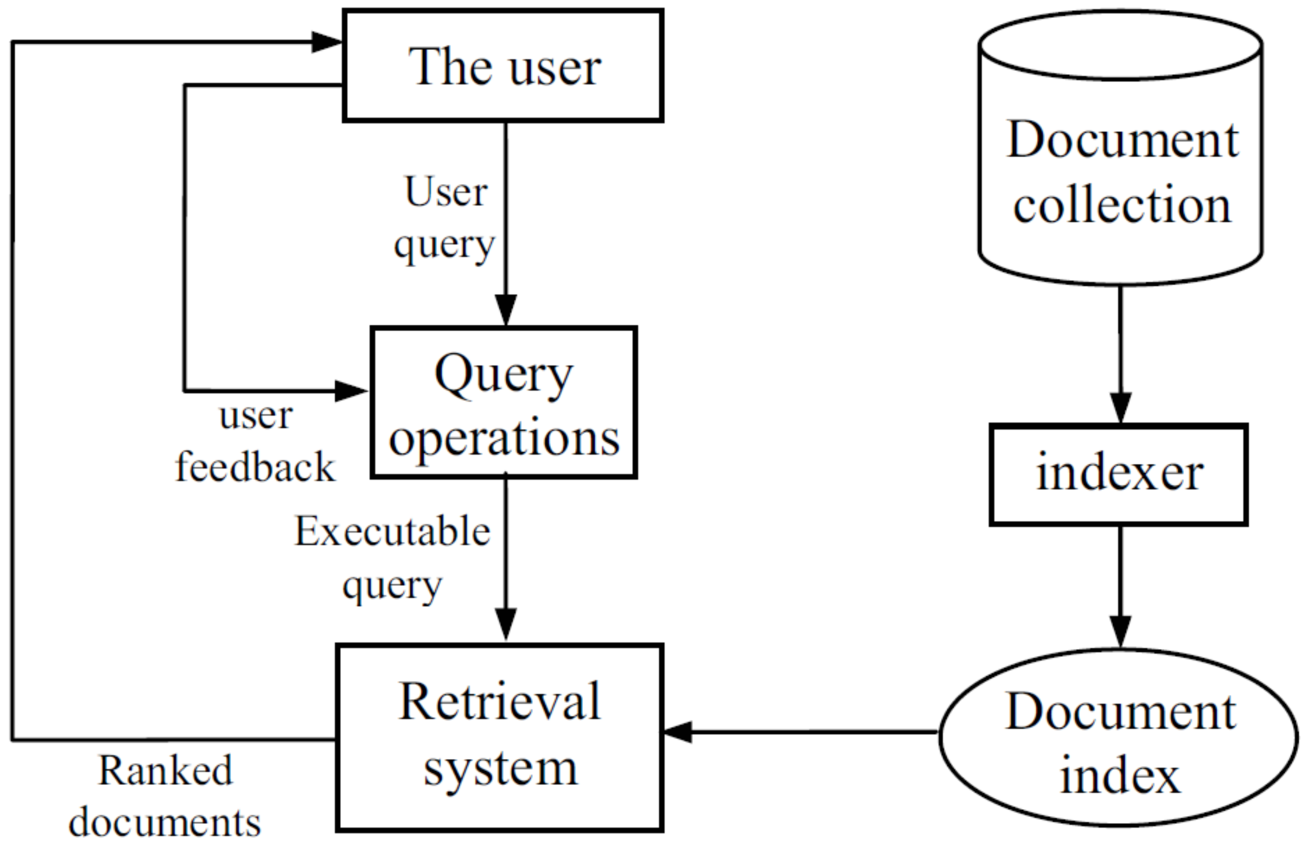
1 BOOK
Web mining : applications and techniques
Hershey, Pa. : Idea Group Publishing , c2005
Available at Au Shue Hung Memorial Library General (006.3 W38 2005)
Online access

2 ARTICLE
Assessing information quality of a learning systems: a web mining approach

Information Retrieval vs Database Search

- Database Search (≠ Information Retrieval)
  - Basic information unit = a data record
  - Highly structured data and stored in relational tables
  - Results are ranked by fields' values
  - Common query format = SQL statement
- What is information retrieval?
  - Basic information unit = a document (unstructured)
  - A large collection of documents = the text "database" (or corpus).
  - Retrieval = finding a set of documents relevant to the user query.
  - The documents are ranked based on relevance scores to the query.
  - Common query format = a list of keywords (also called terms).

The architecture of an IR System



Forms of user queries

1. Keyword queries: list of terms (connected by a "soft" AND). It is not strictly necessary that retrieved document contains all the terms.
2. Boolean queries: terms and Boolean operators (AND, OR, NOT). Only exact matches.
3. Phrase queries: sequence of terms (ex: "web mining techniques")
4. Proximity queries: relaxed phrase queries where documents with query terms within close proximity have higher relevance.
5. Full document queries: the query is a document (ex: similar pages)
6. Natural language questions: the most complex form of query

Components of IR systems

- Query operations module

In the simplest case, it does nothing but just pass the query to the retrieval engine after some simple pre-processing, e.g., removal of stopwords. In more complex cases, it needs to transform natural language queries into executable queries. It may also accept user feedback and use it to expand and refine the original queries (relevance feedback).

- Indexer

It indexes the original raw documents in some data structures to enable efficient retrieval. The result is the document index. The most used data structure is called inverted index (easy to build and efficient to search).

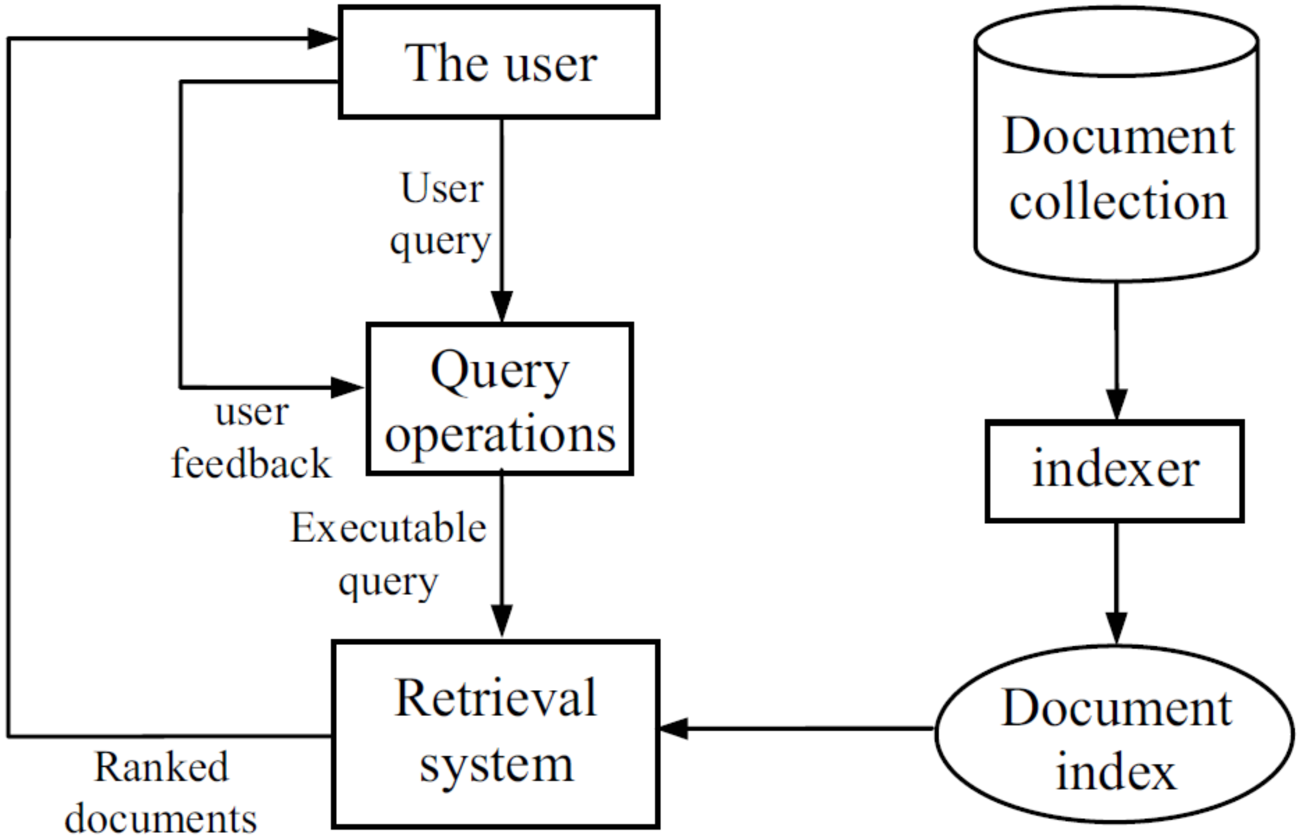
- IR system

It computes a relevance score for each indexed document to the query. According to their relevance scores, the documents are ranked and presented to the user.

Outline

- Goals and architecture of a IR system
- IR models
- Word2Vec and other modern Language Models

Relevance Score



Relevance Score

IR models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined.

- Main models:
  - Boolean model
  - Vector space model
  - Statistical language model

- They all treat each document or query as a “bag” of words or terms.
  - Term sequence and position in a sentence or a document are ignored.
  - Tokenization and other NLP preprocessing usually applies.

Bag of words

- Given a collection of documents D, let $V = \{t_1, t_2, \ldots, t_{|V|}\}$ be the set of distinct words/terms in the collection. V is called the vocabulary.
- A weight $w_{ij} \geq 0$ is associated with each term t_i of a document $\mathbf{d}_j \in D$. For a term that does not appear in document $\mathbf{d}_j$, $w_{ij} = 0$.

$$\mathbf{d}_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j})$$

Document-Term Matrix

- Bag-of-words models make possible to define the so-called Document-Term Matrix

	Word1	Word2	Word3	Word4	Word5	Word6	Word7	Word8	…
Doc1	w_{11}	w_{21}	w_{31}	w_{41}	w_{51}	w_{61}	w_{71}	w_{81}	…
Doc2	w_{12}	w_{22}	w_{32}	w_{42}	w_{52}	w_{62}	w_{72}	w_{82}	…
Doc3	w_{13}	w_{23}	w_{33}	w_{43}	w_{53}	w_{63}	w_{73}	w_{83}	…
Doc4	w_{14}	w_{24}	w_{34}	w_{44}	w_{54}	w_{64}	w_{74}	w_{84}	…
…	…	…	…	…	…	…	…	…	…

Boolean Model

- Document and query are represented by binary vectors whose length is |*V*| and their entries are such that

$$w_{ij} = \begin{cases} 1 & \text{if } t_i \text{ appears in } \mathbf{d}_j \\ 0 & \text{otherwise.} \end{cases}$$

- Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
  - E.g., ((*data* AND *mining*) AND (NOT *text*))
- Exact match: given a Boolean query, the system retrieves every document that makes the query logically true.
- The retrieval results are usually quite poor because term frequency is not considered.

Retrieval in the "only-and" case (1/2)

Documents

- D1 = "Hong Kong is in China"
- D2 = "Hong Kong is a beautiful city"
- D3 = "King Kong is a gorilla"

Preprocessing
(lower-case + stop words removal + lemmatization)

Vocabulary

hong, kong, china, beautiful, city, king, gorilla

Document-Term Matrix

	hong	kong	china	beautiful	city	king	gorilla
D1	1	1	1	0	0	0	0
D2	1	1	0	1	1	0	0
D3	0	1	0	0	0	0	1

Retrieval in the "only-and" case (2/2)

Document-Term Matrix *M*

	hong	kong	china	beautiful	city	king	gorilla
D1	1	1	1	0	0	0	0
D2	1	1	0	1	1	0	0
D3	0	1	0	0	0	0	1

Query

Hong Kong

Preprocessing + Vectorization
(same as before)

t_1=hong t_2=kong

Identify Relevant Documents using Logic

D1 satisfies "t_1 and t_2"
D2 satisties "t_1 and t_2"
D3 does not satisfy "t_1 and t_2"

No rank in the produced results!!!

Vector Space Model

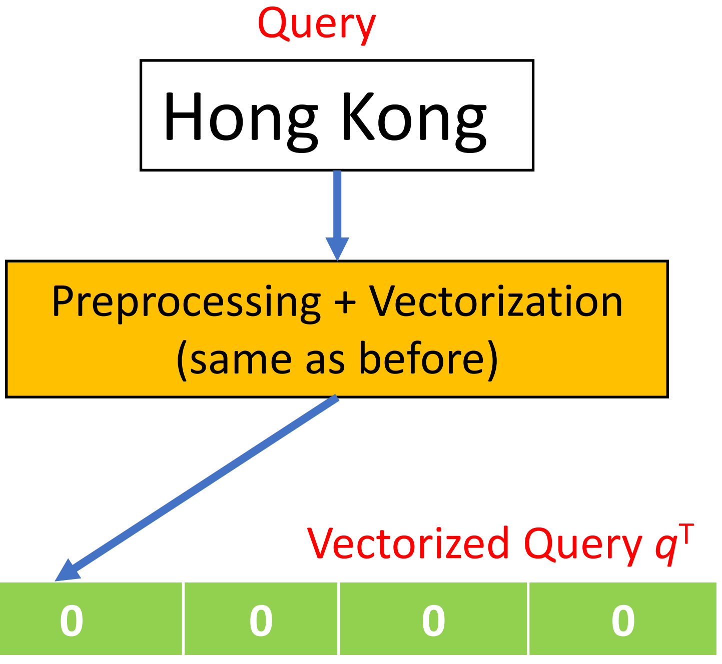
- Documents are also treated as a “bag” of words or terms.
- Each document is represented as a vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.

- Term Frequency (TF) Scheme: The weight of a term t_i in document $\mathbf{d}_j$ is the number of times that t_i appears in $\mathbf{d}_j$, denoted by f_{ij}.
- Normalization may also be applied.

TF: a very first example (using dot-product similarity, not cosine)

Document-Term Matrix *M*

	hong	kong	china	beautiful	city	king	gorilla
D1	1	1	1	0	0	0	0
D2	1	1	0	1	1	0	0
D3	0	1	0	0	0	0	1



Vectorized Query q^T

1	1	0	0	0	0	0

Calculate Relevant Documents

$$Mq =$$

Relevant Documents
2
2
1

Usual matrix-vector multiplication

Vector Space Model

- The most well known weighting scheme
  - TF: still **term frequency**

Normalized TF

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}}$$

  - IDF: **inverse document frequency**.

  N: total number of docs

  df_i: the number of docs where t_i appears.

$$idf_i = \log \frac{N}{df_i}$$

- The final TF-IDF term weight is:

$$w_{ij} = tf_{ij} \times idf_i$$

Retrieval in Vector Space Model

- Query **q** is usually represented in the same way of documents.
- Relevance of $\mathbf{d}_i$ to **q**: Compare the similarity of query **q** and document $\mathbf{d}_i$.
- Cosine similarity (the cosine of the angle between the two vectors)

$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

- Another possibility is to adopt the dot-product as similarity measure
- On the contrary of Boolean model, the similarity allows to rank the documents

Note: Dot product can be used in place of cosine similarity, but the latter remains the preferred score function (recall also that when vectors are normalized, dot product is just a quick way to calculate cosine similarity)

Example: Term Frequency Scheme

d_1	Text analysis is fun. I like doing text analysis.
d_2	I like doing text analysis. I like this. Puppies like this.
d_3	I like puppies, they are fun.
d_4	I like this blog post. This is fun.

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	2	2	1	1	1	1	0	1	0	0	0	0	0
d_2	1	1	0	0	2	3	1	1	0	0	2	0	0
d_3	0	0	0	1	1	1	1	0	1	1	0	0	0
d_4	0	0	1	1	1	1	0	0	0	0	2	1	1

$$d_1 = (2, 2, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0)$$
$$d_2 = (1, 1, 0, 0, 2, 3, 1, 1, 0, 0, 2, 0, 0)$$
$$\ldots$$

Example: TF-IDF (1/3)

Raw Freq.

N=4

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	2	2	1	1	1	1	0	1	0	0	0	0	0
d_2	1	1	0	0	2	3	1	1	0	0	2	0	0
d_3	0	0	0	1	1	1	1	0	1	1	0	0	0
d_4	0	0	1	1	1	1	0	0	0	0	2	1	1

Normalized Term Freq.

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	1	1	0.5	0.5	0.5	0.5	0	0.5	0	0	0	0	0
d_2	0.33	0.33	0	0	0.67	1	0.33	0.33	0	0	0.67	0	0
d_3	0	0	0	1	1	1	1	0	1	1	0	0	0
d_4	0	0	0.5	0.5	0.5	0.5	0	0	0	0	1	0.5	0.5

Document Frequency

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
df_i	2	2	2	3	4	4	2	2	1	1	2	1	1

TF-IDF

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	0.301	0.301	0.151	0.062	0.000	0.000	0.000	0.151	0.000	0.000	0.000	0.000	0.000
d_2	**0.099**	0.099	0.000	0.000	0.000	0.000	0.099	0.099	0.000	0.000	0.202	0.000	0.000
d_3	0.000	0.000	0.000	0.125	0.000	0.000	0.301	0.000	0.602	0.602	0.000	0.000	0.000
d_4	0.000	0.000	0.151	0.062	0.000	0.000	0.000	0.000	0.000	0.000	0.301	0.301	0.301

$$w_{12} = \left(\frac{f_{12}}{max\{f_{12}, f_{22}, \dots, f_{13,2}\}}\right) \times log_{10}\frac{N}{df_1} = (0.33) \times log_{10}\frac{4}{2} = \mathbf{0.099}$$

Example: TF-IDF (2/3)

TF-IDF

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	0.301	0.301	0.151	0.062	0.000	0.000	0.000	0.151	0.000	0.000	0.000	0.000	0.000
d_2	0.099	0.099	0.000	0.000	0.000	0.000	0.099	0.099	0.000	0.000	0.202	0.000	0.000
d_3	0.000	0.000	0.000	0.125	0.000	0.000	0.301	0.000	0.602	0.602	0.000	0.000	0.000
d_4	0.000	0.000	0.151	0.062	0.000	0.000	0.000	0.000	0.000	0.000	0.301	0.301	0.301

Query, $q = \{text, analysis\}$

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
q	0.301	0.301	0	0	0	0	0	0	0	0	0	0	0

$$w_{1q} = \left(\frac{f_{1q}}{max\{f_{1q}, f_{2q}, \ldots, f_{|V|q}\}}\right) \times log_{10}\frac{N}{df_1} = \left(\frac{1}{1}\right) \times log_{10}\frac{4}{2} = 0.301$$

$$w_{2q} = \left(\frac{f_{2q}}{max\{f_{1q}, f_{2q}, \ldots, f_{|V|q}\}}\right) \times log_{10}\frac{N}{df_2} = \left(\frac{1}{1}\right) \times log_{10}\frac{4}{2} = 0.301$$

Example: TF-IDF (3/3)

TF-IDF

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	0.301	0.301	0.151	0.062	0.000	0.000	0.000	0.151	0.000	0.000	0.000	0.000	0.000
d_2	0.099	0.099	0.000	0.000	0.000	0.000	0.099	0.099	0.000	0.000	0.202	0.000	0.000
d_3	0.000	0.000	0.000	0.125	0.000	0.000	0.301	0.000	0.602	0.602	0.000	0.000	0.000
d_4	0.000	0.000	0.151	0.062	0.000	0.000	0.000	0.000	0.000	0.000	0.301	0.301	0.301

Query, $q = \{text, analysis\}$

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
q	0.301	0.301	0	0	0	0	0	0	0	0	0	0	0

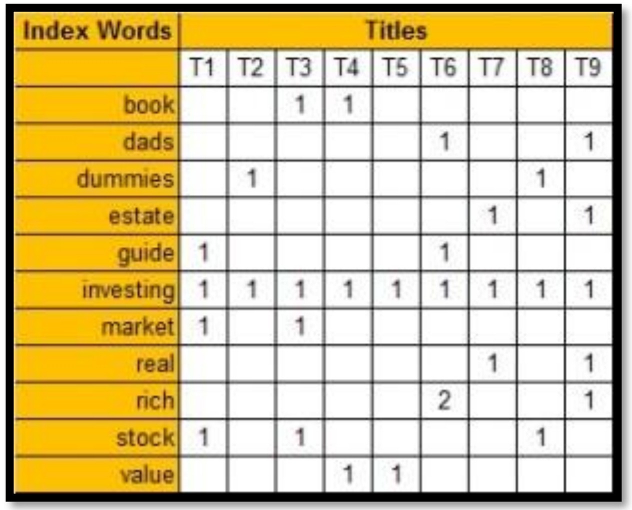
	$d_j \cdot q$	$\cos(d_j, q)$
d_1	0.181	0.886
d_2	0.060	0.495
d_3	0	0
d_4	0	0

Latent Semantic Analysis (LSA)

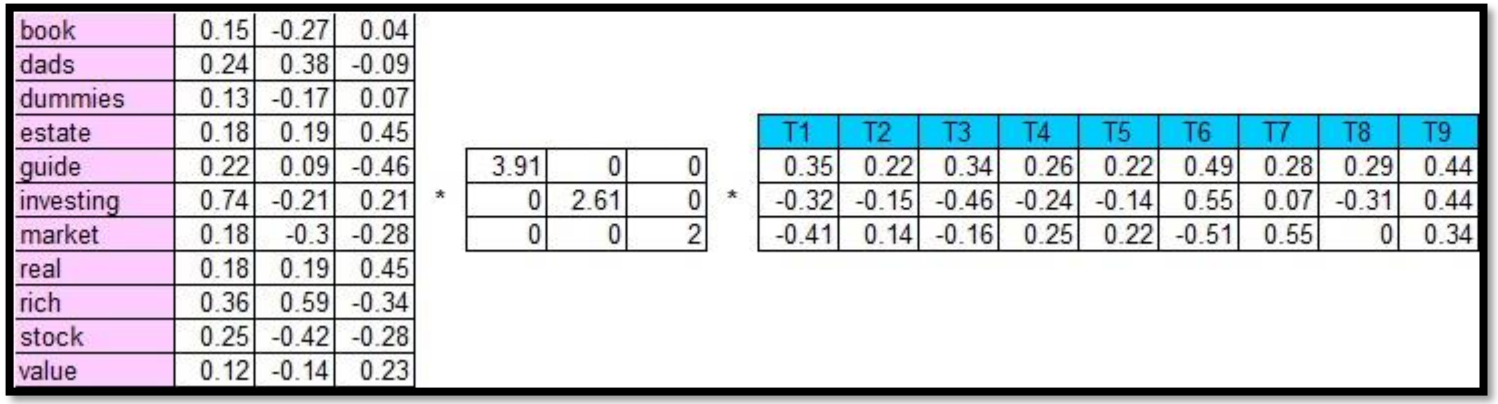
- LSA = Applying Truncated SVD to the document-term matrix

- Motivations:
  - Remove noise
  - Small storage (dimension reduction)
  - Recover missing data via the help of co-occurrence patterns (addressed by the sparsity problem) = synonym words share their weights to each other

Latent Semantic Analysis (LSA)



Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book			1	1					
dads						1			1
dummies		1						1	
estate							1		1
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real							1		1
rich						2			1
stock	1		1					1	
value				1	1				



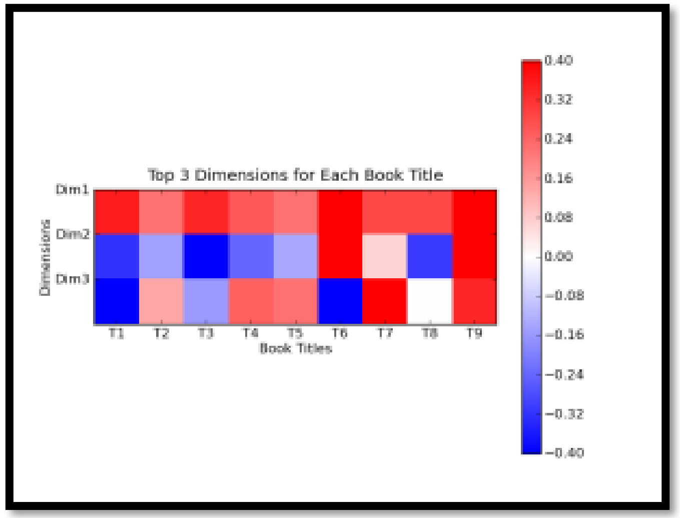
book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.3	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

*

3.91	0	0
0	2.61	0
0	0	2

*

T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0	0.34



https://technowiki.wordpress.com/2011/08/27/latent-semantic-analysis-lsa-tutorial/

Latent Semantic Analysis (LSA)

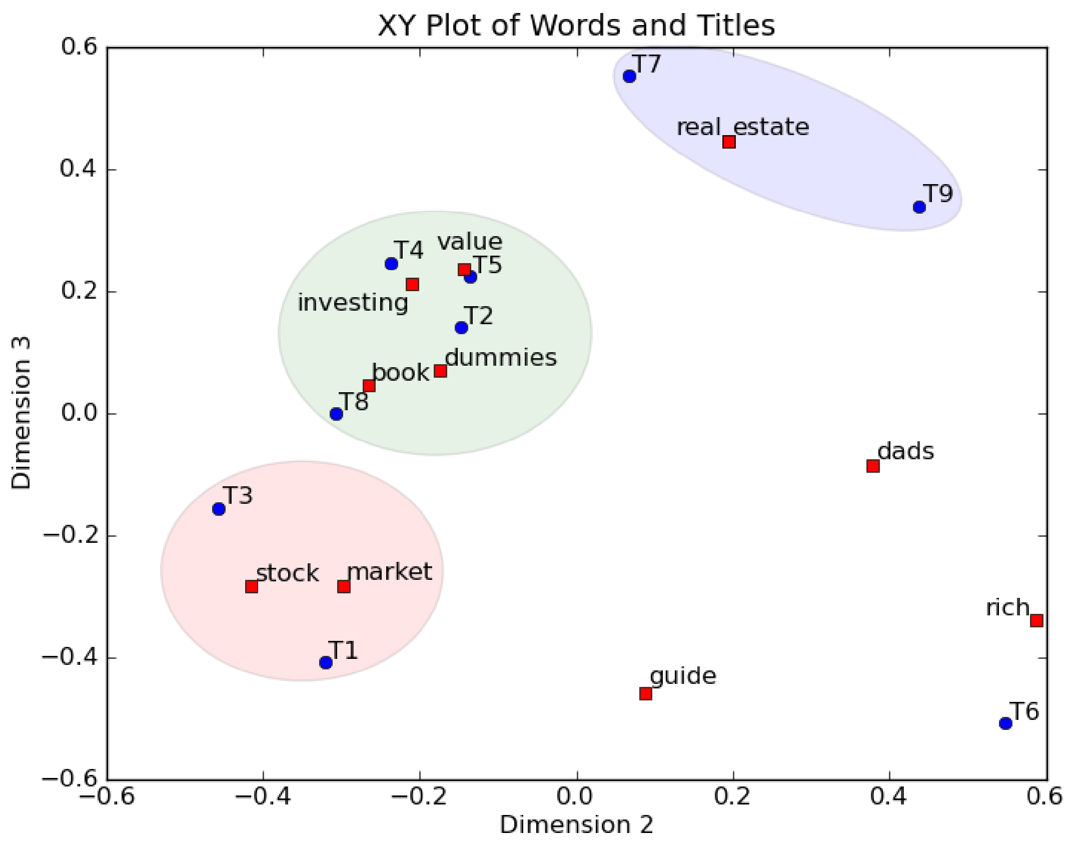
- Document-term matrix $\boldsymbol{X} \in \mathbb{R}^{N \times |\boldsymbol{V}|}$
- Extract latent semantic using Truncated SVD:

$$\boldsymbol{X} \approx \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^{\mathrm{T}}$$

- Document-to-document matrix
  - $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} \approx \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^{\mathrm{T}} \boldsymbol{V}_k \boldsymbol{\Sigma}_k^{\mathrm{T}} \boldsymbol{U}_k^{\mathrm{T}} = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{\Sigma}_k^{\mathrm{T}} \boldsymbol{U}_k^{\mathrm{T}}$
  - document representation: $\boldsymbol{U}_k \in \mathbb{R}^{m \times k}$
- Term-to-term matrix
  - $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \approx \boldsymbol{V}_k \boldsymbol{\Sigma}_k^{\mathrm{T}} \boldsymbol{U}_k^{\mathrm{T}} \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^{\mathrm{T}} = \boldsymbol{V}_k \boldsymbol{\Sigma}_k^{\mathrm{T}} \boldsymbol{\Sigma}_k \boldsymbol{V}_k^{\mathrm{T}}$
  - term representation: $\boldsymbol{V}_k \in \mathbb{R}^{n \times k}$

Semantic Representations in LSA

- Rows of U_k, possibly multiplied by the singular values, are semantic representations of the terms/words
- Rows of V_k (i.e., columns of V_k^T), possibly multiplied by the singular values, are semantic representations of the documents
- Cosine similarity can be calculated on these semantic representations in order to gauge the semantic similarity of terms or documents pairs
- Clustering algorithms can be executed on those representations in order to cluster together similar terms or documents
- Also classification algorithms can trained and executed on those semantic representations



Statistical Language Model

- Let:
  - the query *q* be a sequence of terms, $q = q_1q_2\ldots q_m$
  - the document collection *D* be a set of documents, $D = \{d_1, d_2, \ldots, d_N\}$
- Consider the probability of a query *q* as being "generated" by a probabilistic model based on a document d_j, i.e., $\Pr(q|d_j)$.
- Assuming unigram model, i.e., assume that each word is generated independently (practically, a multinomial distribution over words) so:

$$\Pr(q = q_1q_2\ldots q_m \mid d_j) = \prod_{i=1}^{m} \Pr(q_i \mid d_j) = \prod_{i=1}^{|V|} \Pr(t_i \mid d_j)^{f_{iq}},$$

where f_{iq} is the number of times that term t_i occurs in *q*. Thus $\Pr(t_i \mid d_j) = \frac{f_{ij}}{|d_j|}$.

Retrieval in Statistical Language Model

- To rank documents in retrieval, we are interested in estimating the posterior probability Pr(d_j|q). Using the Bayes rule, we have:

$$\Pr(d_j \mid q) = \frac{\Pr(q \mid d_j)\Pr(d_j)}{\Pr(q)}$$

- Pr(q) is not needed as it is the same for every document
- Pr(d_j) is usually considered uniform and thus will not affect ranking
- We only need to compute the likelihood Pr(q|d_j) as seen before!!!

Example of calculations

$$\prod_{i=1}^{m} \mathrm{P}(q_i|d_j) = \prod_{i=1}^{|V|} \mathrm{P}(t_i|d_j)^{f_{iq}}$$

V = {“I”, “you”, “he”, “she”, “love”, “hate”}

q = {“I”, ”love”, “I”, “love”, “you”}

$\prod_{i=1}^{m} \mathrm{P}(q_i|d_j)$

= P(“I”|d_j) x P(“love”|d_j) x P(“I”|d_j) x P(“love”|d_j) x P(“you”|d_j)

$\prod_{i=1}^{|V|} P(t_i|d_j)^{f_{iq}}$

= P(“I”|d_j)^2 x P(“you”|d_j)^1 x P(“he”|d_j)^0 x P(“she”|d_j)^0 x P(“love”|d_j)^2 x P(“hate”|d_j)^0

Smoothing in Statistical Language Models

- There is practical problem: a query term that does not appear in d_j has the probability of 0, which annihilate Pr(q|d_j).

- Smoothing is adopted to avoid this problem.

$$\Pr_{add}(t_i \mid d_j) = \frac{\lambda + f_{ij}}{\lambda |V| + |d_j|}.$$

  where λ is a very low hyperparameter chosen in [0,1].

- The name smoothing comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward.

Example

$$P(t_i|d_j) = \frac{f_{ij}}{|d_j|}$$

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	0.22	0.22	0.11	0.11	0.11	0.11	0.00	0.11	0.00	0.00	0.00	0.00	0.00
d_2	0.09	0.09	0.00	0.00	0.18	0.27	0.09	0.09	0.00	0.00	0.18	0.00	0.00
d_3	0.00	0.00	0.00	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.00	0.00
d_4	0.00	0.00	0.13	0.13	0.13	0.13	0.00	0.00	0.00	0.00	0.25	0.13	0.13

$$P_{add}(t_i|d_j) = \frac{1 + f_{ij}}{|V| + |d_j|}$$

	text	analysis	is	fun	I	like	puppies	doing	they	are	this	blog	post
d_1	0.14	0.14	0.09	0.09	0.09	0.09	0.05	0.09	0.05	0.05	0.05	0.05	0.05
d_2	0.08	0.08	0.04	0.04	0.13	0.17	0.08	0.08	0.04	0.04	0.13	0.04	0.04
d_3	0.05	0.05	0.05	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.05	0.05	0.05
d_4	0.05	0.05	0.10	0.10	0.10	0.10	0.05	0.05	0.05	0.05	0.14	0.10	0.10

Query, q = “like puppies post”

$$P(q|d_1) = 0.09 \times 0.05 \times 0.05 = 0.000225$$
$$P(q|d_2) = 0.17 \times 0.08 \times 0.04 = 0.000544$$
$$P(q|d_3) = 0.10 \times 0.10 \times 0.05 = 0.0005$$
$$P(q|d_4) = 0.10 \times 0.05 \times 0.10 = 0.0005$$

$$-\log P(q|d_1) = 3.648$$
$$-\log P(q|d_2) = \mathbf{3.264}$$
$$-\log P(q|d_3) = 3.301$$
$$-\log P(q|d_4) = 3.301$$

Minimize negative-log is just a trick that makes numbers "readable". Log-likelihood is also more efficient for calculations (sums in place of products)

Limitation of the bag-of-words approaches

- Will the word ordering in the document affect the document representation?
- Yes!
- Considering *n*-grams allows to overcome this limitation …
  - Example of bigrams (or 2-grams):

    The text "Hong Kong is a beatufiul city" contains the following bigrams:

    Hong_Kong, Kong_is, is_a, a_beautiful, beautiful_city
- … but it is computationally more expensive … much more if *n*>2

Outline

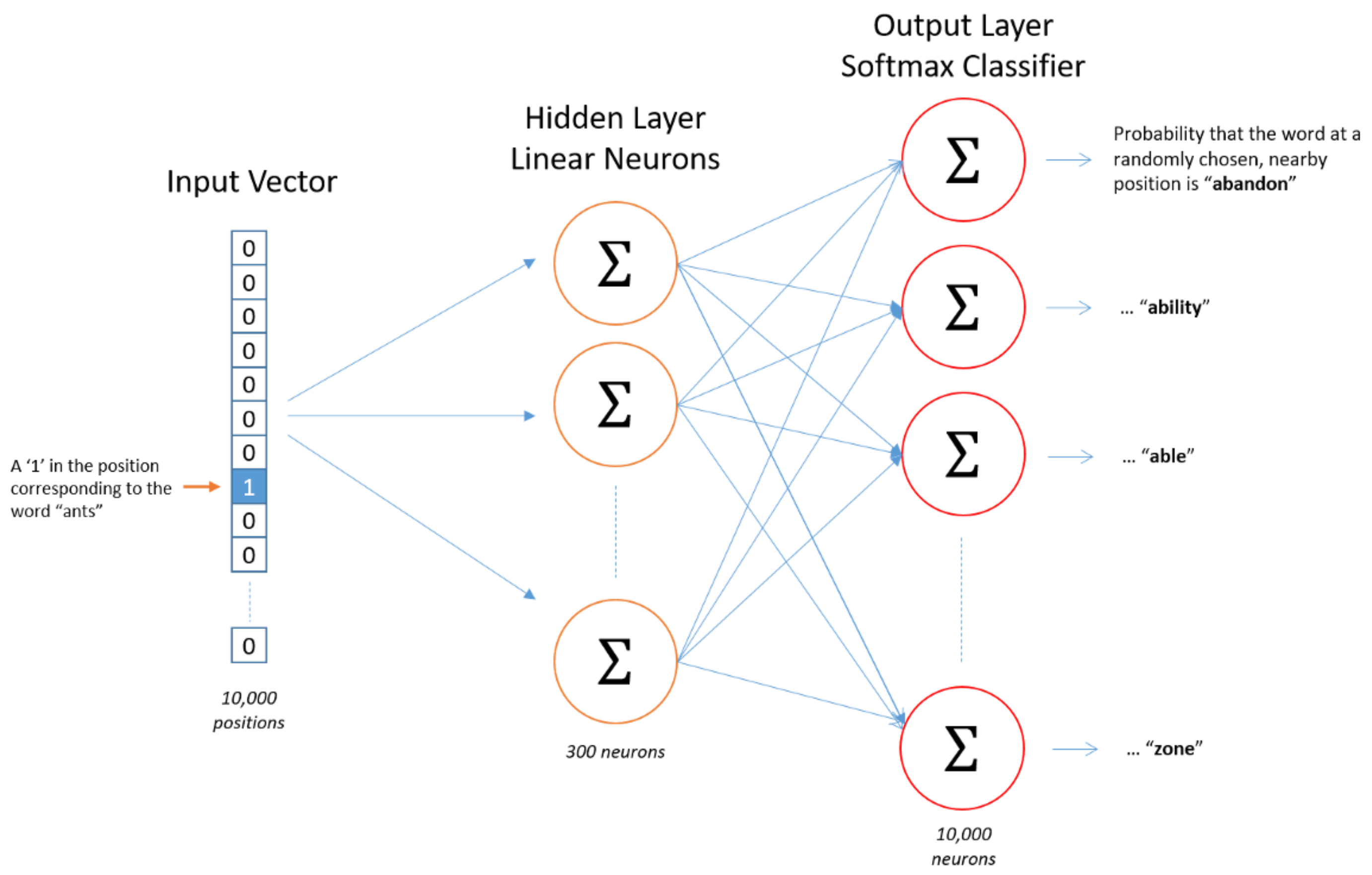
- Goals and architecture of a IR system
- IR models
- <u>Word2Vec and other modern Language Models</u>

Vector representation used in Spacy?

- It uses Word2Vec, which generates a vector for every word
- Word2vec explores the term co-occurrence in a local neighborhood (nearby words, and also called context words).
- E.g., size of neighbourhood =2
  - "I like this blog post. This is fun.".
  - Center word: brown (also called focus word).
  - Context words: blue (also called target words).
- Word2Vec considers all words as center words, and all their context words so that words can be predicted!
- Linguistic principle: two words W and V are semantically related if they appear in similar contexts throughout a (very large) corpus of texts.

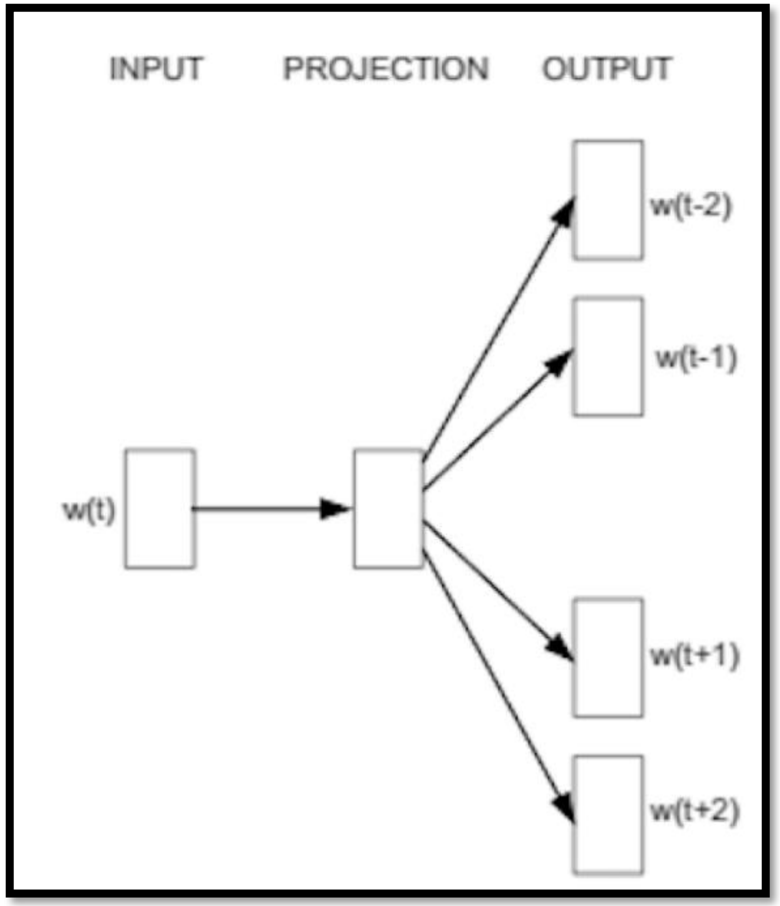
Word2Vec (Skipgram)

- It uses an AUTO-ENCODER-like neural network trained on a large corpus

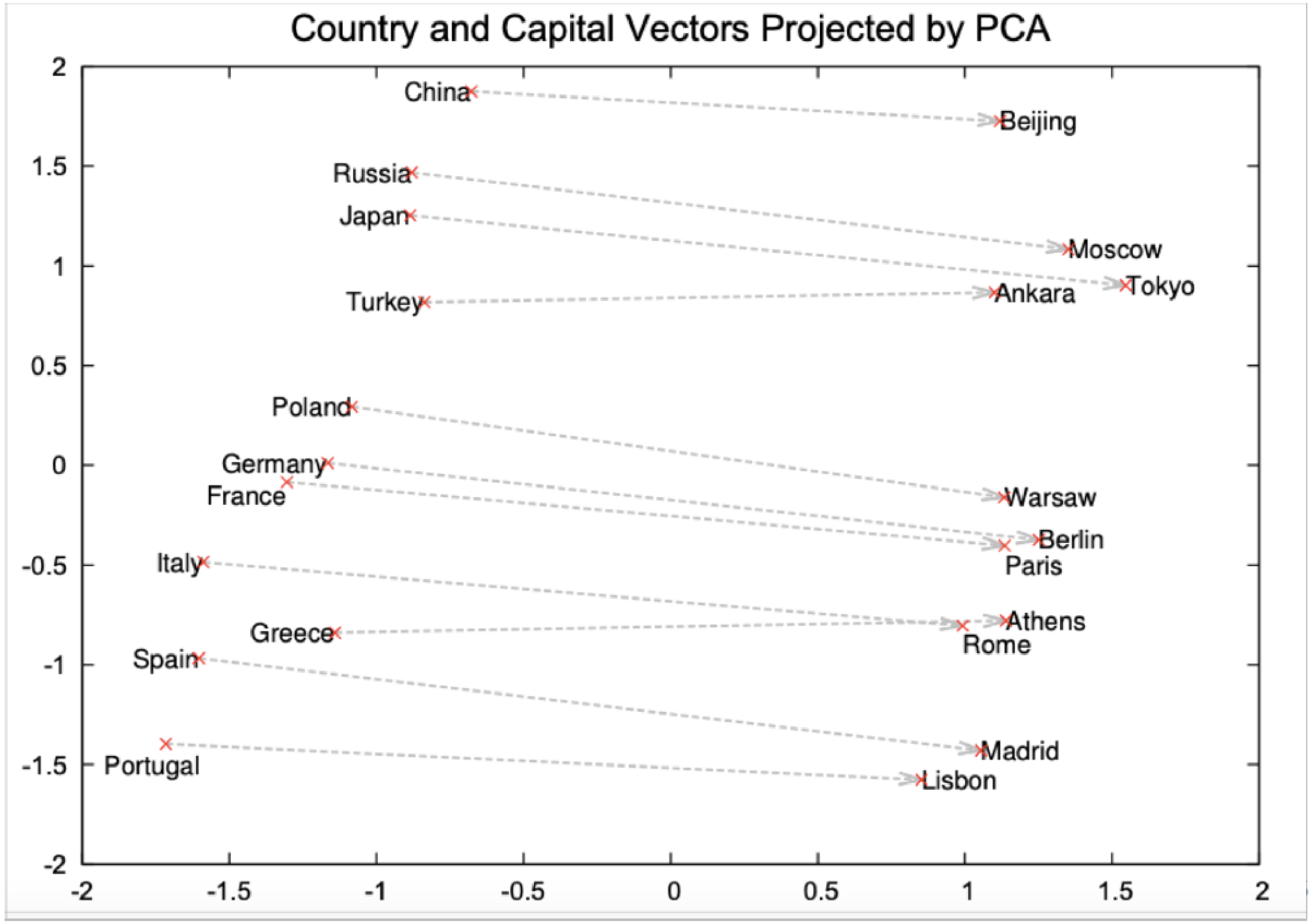


Word2Vec (Skipgram)

- Maximize the joint probability of predicting context words $w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m}$ given w_t
- $J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log p\left(w_{t+j} \middle| w_t\right)$
- $p\left(w_{t+j} \middle| w_t\right) = \frac{\exp(u_{w_{t+j}} \cdot v_{w_t})}{\sum_{w'} \exp(u_{w'} \cdot v_{w_t})}$
- Every word has 2 vectors
  - v_w : when w is the center word
  - u_w : when w is the context word
- Obtain the representation by maximization



Word2Vec: Analogies emerge after learning



From Word2Vec to Sentence Embeddings?

- Word vectors generated by Word2Vec can be aggregated to have a document vector
- But … again word ordering does not affect the document vector

- Very recent Large Language Model based on neural networks allow to generate semantically meaningful document embedding by allowing the calculation of "contextual word embeddings" (same words has different embeddings based on the context where it appears) that can be "safely" aggregated to form semantically meaningful sentence embeddings
- Example:
  - SBERT =
  - = Sentence BERT =
  - = Sentence Bidirectional Encoder Representations from Transformers
  - https://www.sbert.net/

References

- *Liu, Bing. Web data mining: exploring hyperlinks, contents, and usage data. Berlin: springer, 2011. Chapter 6.*

- *Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality," In Proceedings of NIPS 2013, pp. 3111-3119*

- https://www.sbert.net/