Information Retrieval
Latent Semantic Analysis
(SNLP tutorial)

Vilém Zouhar

March 12, 2021

Overview

- Information retrieval
- \- Metrics
- \- Preprocessing
- Retrieval using LM
- Retrieval example
- Document vector representation
- \- Solution 1 (counts)
- \- Solution 2 (tf)
- \- Solution 3 (tf-idf)
- \- Solution 4 (LSA, SVD)
- Code & Considerations
- Homework

Information retrieval - metrics

- Documents D, queries Q
- System: $Q \rightarrow \mathcal{P}(D)$
- For $q \in Q$: retrieved (output), relevant (gold)

Information retrieval - metrics

- Documents D, queries Q
- System: $Q \rightarrow \mathcal{P}(D)$
- For $q \in Q$: retrieved (output), relevant (gold)
- Recall $\frac{|\text{retrieved} \cap \text{relevant}|}{|\text{relevant}|}$
- Precision $\frac{|\text{retrieved} \cap \text{relevant}|}{|\text{retrieved}|}$

Information retrieval - metrics

- Documents D, queries Q
- System: $Q \to \mathcal{P}(D)$
- For $q \in Q$: retrieved (output), relevant (gold)
- Recall $\frac{|\text{retrieved} \cap \text{relevant}|}{|\text{relevant}|}$
- Precision $\frac{|\text{retrieved} \cap \text{relevant}|}{|\text{retrieved}|}$
- System: $Q \times D \to \mathbb{R}$
- $\{$Precision,Recall$\}$@k retrieve k documents (top k scoring)
- Recall@k $\frac{|\text{retrieved@}k \cap \text{relevant}|}{|\text{relevant}|}$
- Precision@k $\frac{|\text{retrieved@}k \cap \text{relevant}|}{k}$

Information retrieval - metrics

- Average precision: $AveP(q) = \frac{\sum_1^n P@k \times rel(k)}{|\text{relevant}|}$
- $rel(k) = \begin{cases} 1 & k\text{-th document relevant} \\ 0 & otherwise \end{cases}$

Information retrieval - metrics

- Average precision: $AveP(q) = \frac{\sum_1^n P@k \times rel(k)}{|\text{relevant}|}$
- $rel(k) = \begin{cases} 1 & k\text{-th document relevant} \\ 0 & otherwise \end{cases}$
- Mean average precision $MAP(Q) = \frac{\sum_{q \in Q} AveP(q)}{|Q|}$
-
  - Q can be a "testset"

Information retrieval - metrics

- Average precision: $AveP(q) = \frac{\sum_1^n P@k \times rel(k)}{|\text{relevant}|}$
- $rel(k) = \begin{cases} 1 & k\text{-th document relevant} \\ 0 & otherwise \end{cases}$
- Mean average precision $MAP(Q) = \frac{\sum_{q \in Q} AveP(q)}{|Q|}$
-
    - Q can be a "testset"
- F-score $2 \cdot \frac{p \cdot r}{p + r}$
- F-score@k $2 \cdot \frac{p@k \cdot r@k}{p@k + r@k} = 2 \cdot \frac{p@k \cdot r@k}{k + r@k}$

Information retrieval - metrics

- Taking the rank into consideration
- Mean Reciprocal Rank
- rank_q = position of the first relevant document
- $MRR(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$

Information retrieval - metrics

- Taking the rank into consideration
- Mean Reciprocal Rank
- rank_q = position of the first relevant document
- $MRR(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$

document	position	relevant
a	4	+
b	1	
c		
d		+
e	2	+
f	3	

Information retrieval - metrics

- Taking the rank into consideration
- Mean Reciprocal Rank
- rank_q = position of the first relevant document
- $MRR(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$

document	position	relevant
a	4	+
b	1	
c		
d		+
e	2	+
f	3	

- $Q = \{\text{example}\}, MRR(Q) = \frac{1}{\text{rank}_{example}} = \frac{1}{2}$

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- \- Not always: query *becomes stressed* vs. *becom stress*

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)

Information retriveal - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*

Information retriveal - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)

Information retriveal - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)
- - Not always: query *Wizard of Oz* vs. *Wizard Oz*

Information retrieval - preprocessing

- Stemming (*going* → *go*, *studies* → *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* → *go*, *studies* → *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)
- - Not always: query *Wizard of Oz* vs. *Wizard Oz*
- Typo correction (*Wizzard* → *Wizard*)

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)
- - Not always: query *Wizard of Oz* vs. *Wizard Oz*
- Typo correction (*Wizzard* $\rightarrow$ *Wizard*)
- - Not always: query *Tokyo* vs. *Tokio*

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)
- - Not always: query *Wizard of Oz* vs. *Wizard Oz*
- Typo correction (*Wizzard* $\rightarrow$ *Wizard*)
- - Not always: query *Tokyo* vs. *Tokio*

Information retrieval - preprocessing

- Stemming (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *studi*)
- - Not always: query *becomes stressed* vs. *becom stress*
- Lemmatization (*going* $\rightarrow$ *go*, *studies* $\rightarrow$ *study*)
- - Not always: query *becomes stressed* vs. *become stress*
- Stop words (*for, of, and, or*)
- - Not always: query *Wizard of Oz* vs. *Wizard Oz*
- Typo correction (*Wizzard* $\rightarrow$ *Wizard*)
- - Not always: query *Tokyo* vs. *Tokio*

Always depends on the task.

Document retrieval - example

- Query: **Goethe, devil**

Document retrieval - example

- Query: **Goethe, devil**
- Document:

  A: Wolfgang's idea of the demon Mephistopheles who makes a bet with God

  B: Faust is Wolfgang **Goethe**'s play in German about a pact with the **devil**

  C: **Devil**ishly good lasagne

  D: The impact of **Goethe**'s demon play on the German literature:

Document retrieval - example

- Query: **Goethe, devil**
- Document:

  A: Wolfgang's idea of the demon Mephistopheles who makes a bet with God

  B: Faust is Wolfgang **Goethe**'s play in German about a pact with the **devil**

  C: **Devil**ishly good lasagne

  D: The impact of **Goethe**'s demon play on the German literature:
- How to rank them?

  B (contains the two key words)

  D (Goethe, literature)

  A (Wolfgang - Goethe, Mephistopheles - devil)

  C (unrelated context)

Document retrieval - example

- Query: **Goethe, devil**
- Document:
  A: Wolfgang's idea of the demon Mephistopheles who makes a bet with God
  B: Faust is Wolfgang **Goethe**'s play in German about a pact with the **devil**
  C: **Devil**ishly good lasagne
  D: The impact of **Goethe**'s demon play on the German literature:
- How to rank them?
  B (contains the two key words)
  D (Goethe, literature)
  A (Wolfgang - Goethe, Mephistopheles - devil)
  C (unrelated context)
- Can these inferences be made automatically? [2]

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \ p(d|q) = argmax_d \ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \ p(q|d) \cdot p(d)$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \ p(d|q) = argmax_d \ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \ p(q|d) \cdot p(d)$
- $\approx argmax_d \ p_{LM}(q|d) \cdot p(d)$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \; p(d|q) = argmax_d \; \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \; p(q|d) \cdot p(d)$
- $\approx argmax_d \; p_{LM}(q|d) \cdot p(d)$
- $p(d) \approx \frac{1}{|D|}$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \ p(d|q) = argmax_d \ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \ p(q|d) \cdot p(d)$
- $\approx argmax_d \ p_{LM}(q|d) \cdot p(d)$
- $\qquad p(d) \approx \frac{1}{|D|}$
- $\approx argmax_d \ p_{LM}(q|d)$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \ p(d|q) = argmax_d \ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \ p(q|d) \cdot p(d)$
- $\approx argmax_d \ p_{LM}(q|d) \cdot p(d)$
- $\qquad p(d) \approx \frac{1}{|D|}$
- $\approx argmax_d \ p_{LM}(q|d)$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \; p(d|q) = argmax_d \; \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \; p(q|d) \cdot p(d)$
- $\approx argmax_d \; p_{LM}(q|d) \cdot p(d)$
- $\quad p(d) \approx \frac{1}{|D|}$
- $\approx argmax_d \; p_{LM}(q|d)$
- Unigram: $p(d|q) \approx \prod_i p_{LM}(q_i|d)$

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d \ p(d|q) = argmax_d \ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d \ p(q|d) \cdot p(d)$
- $\approx argmax_d \ p_{LM}(q|d) \cdot p(d)$
- $\qquad p(d) \approx \frac{1}{|D|}$
- $\approx argmax_d \ p_{LM}(q|d)$
- Unigram: $p(d|q) \approx \prod_i p_{LM}(q_i|d)$
- Jelinek-Mercer smoothing [9]: $p(q_i|d, D) = \lambda \cdot p(q_i|d) + (1 - \lambda) \cdot p(q_i|D)$
- High λ: documents with all query words (conjunctive)
- Low λ: suitable for long queries (disjunctive)

Document retrieval - Language Model

- Pretend the query was generated by a LM based on the document
- $argmax_d\ p(d|q) = argmax_d\ \frac{p(q|d) \cdot p(d)}{p(q)} = argmax_d\ p(q|d) \cdot p(d)$
- $\approx argmax_d\ p_{LM}(q|d) \cdot p(d)$
- $\quad p(d) \approx \frac{1}{|D|}$
- $\approx argmax_d\ p_{LM}(q|d)$

- Unigram: $p(d|q) \approx \prod_i p_{LM}(q_i|d)$

- Jelinek-Mercer smoothing [9]: $p(q_i|d, D) = \lambda \cdot p(q_i|d) + (1 - \lambda) \cdot p(q_i|D)$
- High λ: documents with all query words (conjunctive)
- Low λ: suitable for long queries (disjunctive)

- Issue: Without word embeddings, no word relatedness
  Query: **Goethe, devil**
  A: Wolfgang's idea of the demon Mephistopheles who makes a bet with God

Document vector representation

- Represent the query and all documents as a vector
  Measure their similarity (L-norm, cosine distance: $\frac{D \cdot Q}{|D||Q|}$)

Document vector representation

- Represent the query and all documents as a vector
  Measure their similarity (L-norm, cosine distance: $\frac{D \cdot Q}{|D||Q|}$)
- How to represent a query/document as a fixed size vector?

Solution 1 (counts)

- Solution: vector with counts of words:
  `(<the>, <a>, <dog>, <president>, ...)`
  `(57, 68, 0, 2, ...)`

Solution 1 (counts)

- Solution: vector with counts of words:
  `(<the>, <a>, <dog>, <president>, ...)`
  `(57, 68, 0, 2, ...)`
- Issue: representation vectors are enormous

Solution 1 (counts)

- Solution: vector with counts of words:
  `(<the>, <a>, <dog>, <president>, ...)`
  `(57, 68, 0, 2, ...)`
- Issue: representation vectors are enormous
- Issue: longer documents have naturally higher counts

Solution 1 (counts)

- Solution: vector with counts of words:
  `(<the>, <a>, <dog>, <president>, ...)`
  `(57, 68, 0, 2, ...)`
- Issue: representation vectors are enormous
- Issue: longer documents have naturally higher counts
- Issue: useless stop words

Solution 2 (tf)

- Solution: vector with counts of non-stop words, normalized by total words:
  (<dog>, <president>, <princess>, <thing>, ...)
  (0, 0.0003, 0.00001, 0.08, ...).

Solution 2 (tf)

- Solution: vector with counts of non-stop words, normalized by total words:
  `(<dog>, <president>, <princess>, <thing>, ...)`
  `(0, 0.0003, 0.00001, 0.08, ...)`.
- Issue: some words naturally occur with higher frequency but don't contribute to document meaning (`<thing>`)

Solution 2 (tf)

- Solution: vector with counts of non-stop words, normalized by total words:
  `(<dog>, <president>, <princess>, <thing>, ...)`
  `(0, 0.0003, 0.00001, 0.08, ...).`
- Issue: some words naturally occur with higher frequency but don't contribute to document meaning (`<thing>`)
- Issue: how do we know which words are useful?

Term Frequency - Inverse Document Frequency

TF-IDF

$$tf(term, doc) = \frac{count_{doc}(term)}{|doc|}$$

$$df(term) = \frac{|\{doc|term \in doc, doc \in D\}|}{|D|}$$

$$idf'(term) = \frac{|D|}{df(term)}, idf(term) = \log_2\left(\frac{|D|}{df(term)}\right)$$

$$tf - idf(term, doc) = tf(term, doc) \times idf(term)$$

Term Frequency - Inverse Document Frequency

TF-IDF

$$tf(term, doc) = \frac{count_{doc}(term)}{|doc|}$$

$$df(term) = \frac{|\{doc|term \in doc, doc \in D\}|}{|D|}$$

$$idf'(term) = \frac{|D|}{df(term)}, idf(term) = \log_2\left(\frac{|D|}{df(term)}\right)$$

$$tf - idf(term, doc) = tf(term, doc) \times idf(term)$$

Augmented TF

$$tf'(term, doc) = 0.5 + 0.5 \cdot \frac{count_{doc}(term)}{max_{term'}\{count_{doc}(term')\}}$$

Solution 3

- Solution: vector of tf-idf
- Good metrics to determine the significance of a term in a document collection
- Issue: still enormous vectors

Solution 3

- Solution: vector of tf-idf
- Good metrics to determine the significance of a term in a document collection

- Issue: still enormous vectors
- Issue: `demon - Mephistopheles` are equally separate concepts as `demon - lassagne`

Solution 3

- Solution: vector of tf-idf
- Good metrics to determine the significance of a term in a document collection

- Issue: still enormous vectors
- Issue: `demon - Mephistopheles` are equally separate concepts as `demon - lassagne`
- Issue: independent terms assumption

Solution 4 (LSA)

- Solution: Perform dimensionality reduction using SVD
- $\rightarrow$ eigenvalues, singular value decomposition
- $A_{i,j} = \#$ occurences of term t_i id document d_j (replace with tf-idf later)

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
Faust	0	1	0	0
Goethe	0	1	0	1
devil	0	1	1	0
demon	1	0	0	1
lassagne	0	0	1	0
German	0	1	0	1

Approximation of A

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	**1**	0	**1**
Faust	**1**	1	0	0
Goethe	**1**	1	0	**0**
devil	**1**	1	**0**	**1**
demon	1	**1**	0	1
lassagne	0	0	1	0
German	**1**	1	0	**0**

	c_1	c_2	c_3
Wolfgang	1	0	0
Mephistopheles	0	1	0
Faust	1	0	0
Goethe	1	0	0
devil	0	1	0
demon	0	1	0
lassagne	0	0	1
German	1	0	0

3 latent concepts:
{Goethe (Wolfgang, Faust, German), devil (Mephistopheles, demon), lassagne}

Approximation of A

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	**1**	0	**1**
Faust	**1**	1	0	0
Goethe	**1**	1	0	**0**
devil	**1**	1	**0**	**1**
demon	1	**1**	0	1
lassagne	0	0	1	0
German	**1**	1	0	**0**

	c_1	c_2	c_3
Wolfgang	1	0	0
Mephistopheles	0	1	0
Faust	1	0	0
Goethe	1	0	0
devil	0	1	0
demon	0	1	0
lassagne	0	0	1
German	1	0	0

3 latent concepts:
{Goethe (Wolfgang, Faust, German), devil (Mephistopheles, demon), lassagne}

$$d_1 = 1 \times c_1 + 1 \times c_2$$

Approximation of A

- Given: A, k

Approximation of A

- Given: A, k
- $A' = argmin_{A' \text{rank} k} ||A - A'||$

Approximation of A

- Given: A, k
- $A' = argmin_{A' \text{rank} k} ||A - A'||$
- Distance e.g. Frobenius ($\sqrt{\sum_{i,j} a_{i,j}}$)

SVD

- $A_{i,j} = \#$ occurences of term t_i id document d_j (replace with tf-idf later)
- $(A^T A)_{i,j} = \#$ intersection of documents d_i and d_j
- $(AA^T)_{i,j} = \#$ documents in which both terms t_i and t_j occur (multiplied counts)
- $U =$ eigenvectors of $A^T A$
- $V =$ eigenvectors of AA^T
- $S =$ roots of corresponding eigenvalues of $A^T A$
- $A = USV^T$

Eigen{vector,value}

Nonzero $v \in \mathbb{R}^n, \lambda \in \mathbb{R}$

Eigenvector

$$Av = \lambda v \qquad Av = \lambda I v \qquad (A - \lambda I)v = 0 \qquad ker(A - \lambda I)$$

"Directions (v) which A only scales."

Eigenvalue

$$Av = \lambda v$$

"The stretch (λ) of eigenvector v by A."

SVD

Proof sketch

$$A = USV^T, A^T = VSU^T, S \text{ diagonal}$$

$$U^T U = VV^T = I \text{ orthogonal}$$

$$AA^T U = US^2 \rightarrow U \text{ eigenvectors of } AA^T, S \text{ root of eigenvalues}$$

$$(\forall i : AA^T U_{i,*} = U_{i,*} \cdot S_{i,i}^2)$$

$$A^T AV = VS^2 \rightarrow V \text{ eigenvectors of } A^T A, S \text{ root of eigenvalues}$$

$$(\forall i : A^T AV_{i,*} = V_{i,*} \cdot S_{i,i}^2)$$

LSA

1. Order eigenvalues by descending values ($S_{i,i} > S_{i+1,i+1} \geq 0$) (proof next slide)
2. Take top-k eigenvectors + values (or all above threshold)
3. $A_K = U_K S_K V_K^T$ $[(m \times n), (n \times n), (n \times n)] \rightarrow [(m \times k), (k \times k), (k \times n)]$

LSA

1. Order eigenvalues by descending values ($S_{i,i} > S_{i+1,i+1} \geq 0$) (proof next slide)
2. Take top-k eigenvectors + values (or all above threshold)
3. $A_K = U_K S_K V_K^T \; [(m \times n), (n \times n), (n \times n)] \rightarrow [(m \times k), (k \times k), (k \times n)]$

- Term $\rightarrow$ latent representation: $U_k S_k$

LSA

1. Order eigenvalues by descending values ($S_{i,i} > S_{i+1,i+1} \geq 0$) (proof next slide)
2. Take top-k eigenvectors + values (or all above threshold)
3. $A_K = U_K S_K V_K^T$ $[(m \times n), (n \times n), (n \times n)] \rightarrow [(m \times k), (k \times k), (k \times n)]$

- Term $\rightarrow$ latent representation: $U_k S_k$
- Document $\rightarrow$ latent representation: $(S_k V_k^T)^T = V_k S_k^T = V_k S_k$

Properties of S

Descending

$$U' = U \text{ +swapped } i,j \text{ column}, S' = S \text{ +swapped } i,j \text{ values}, {V'}^T = V^T \text{ +swapped } i,j \text{ row}$$

$$U' = U \times C(i,j), S' = S \times C(i,j), {V'}^T = V^T \times R(i,j)$$

$$U'S' = (US) \text{ with swapped } i,j \text{ columns}, U'S' = (US) \times C(i,j)$$

$$U'S'{V'}^T = (US) \times C(i,j) \times V^T \times R(i,j) = (US) \times C(i,j) \times C(i,j) V^T = USV^T$$

Properties of S

Descending

$$U' = U \text{ +swapped } i,j \text{ column}, S' = S \text{ +swapped } i,j \text{ values}, {V'}^T = V^T \text{ +swapped } i,j \text{ row}$$

$$U' = U \times C(i,j), S' = S \times C(i,j), {V'}^T = V^T \times R(i,j)$$

$$U'S' = (US) \text{ with swapped } i,j \text{ columns}, U'S' = (US) \times C(i,j)$$

$$U'S'{V'}^T = (US) \times C(i,j) \times V^T \times R(i,j) = (US) \times C(i,j) \times C(i,j)V^T = USV^T$$

Non-negative

$$A^T A \text{ is positive semidefinite} \Rightarrow S_{i,i} \geq 0$$

$$\forall x \neq \overrightarrow{0} : x^T A^T Ax = (Ax)^T (Ax) = ||Ax|| \geq 0$$

LSA Concepts

- $U_k S_k$ maps terms to latent "concepts" $(m \rightarrow k)$
- $V_k S_k$ maps documents to "concepts" $(n \rightarrow k)$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$
- Representation of `Goethe`: fourth row of U_k $(m \times k \rightarrow 1 \times 2)$ scaled by S_k: $[0.13, -0.13]^T$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$
- Representation of `Goethe`: fourth row of U_k $(m \times k \rightarrow 1 \times 2)$ scaled by S_k: $[0.13, -0.13]^T$
- Representation of `devil`: fifth row of U_k $(m \times k \rightarrow 1 \times 2)$ scaled by S_k: $[0.58, -0.01]^T$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$
- Representation of `Goethe`: fourth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.13, -0.13]^T$
- Representation of `devil`: fifth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.58, -0.01]^T$
- Representation of d_1: first column of V_k^T $(k \times n \to 2 \times 1)$ scaled first by S_k: $r_d = [0.3, 0.02]^T$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$
- Representation of `Goethe`: fourth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.13, -0.13]^T$
- Representation of `devil`: fifth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.58, -0.01]^T$
- Representation of d_1: first column of V_k^T $(k \times n \to 2 \times 1)$ scaled first by S_k:
  $r_d = [0.3, 0.02]^T$
- Query representation: vector average:
  $r_q = [0.13, -0.13]^T/2 + [0.58, -0.01]^T/2 = [0.355, -0.07]^T$

LSA Example

	d_1	d_2	d_3	d_4
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
...				

- Choose $k = 2$
- Representation of `Goethe`: fourth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.13, -0.13]^T$
- Representation of `devil`: fifth row of U_k $(m \times k \to 1 \times 2)$ scaled by S_k: $[0.58, -0.01]^T$
- Representation of d_1: first column of V_k^T $(k \times n \to 2 \times 1)$ scaled first by S_k: $r_d = [0.3, 0.02]^T$
- Query representation: vector average: $r_q = [0.13, -0.13]^T/2 + [0.58, -0.01]^T/2 = [0.355, -0.07]^T$
- Query-document match: cosine similairty: $\frac{r_q \cdot r_d}{|r_q| \cdot |r_d|} = \frac{0.01205}{0.10879} \approx 0.11$

LSA Graphics

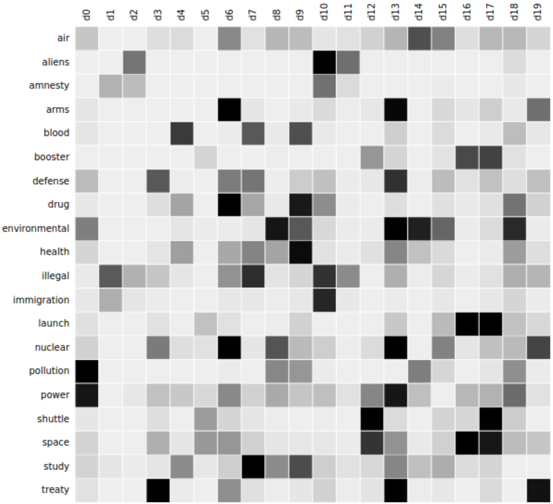

Figure 1: Term-document matrix, no ordering, $k = 5$; Source [6]

LSA Graphics

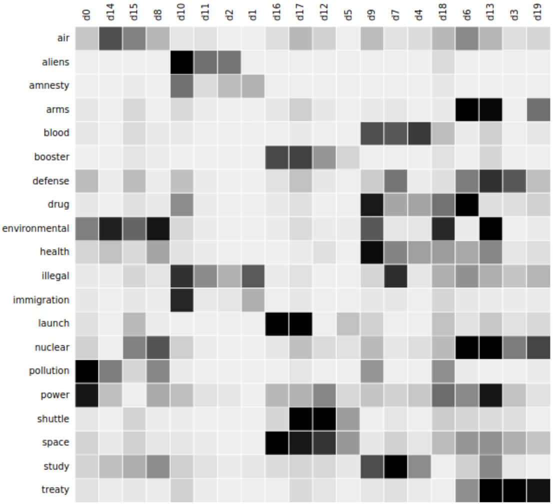

Figure 2: Term-document matrix, group documents, $k = 5$; Source [6]

LSA Graphics

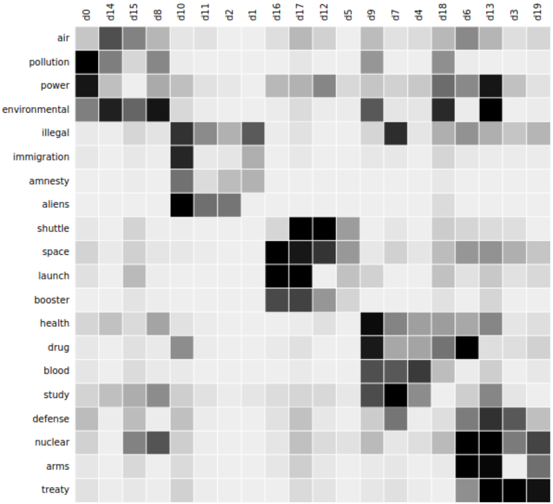

Figure 3: Term-document matrix, group documents+terms, $k = 5$; Source [6]

LSA Code

```
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english',
    max_features= 1000,
    max_df = 0.5,
    smooth_idf=True)
X = vectorizer.fit_transform(documents)

svd_model = TruncatedSVD(n_components=20)
svd_model.fit(X)
```

LSA Code

```
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english',
    max_features= 1000,
    max_df = 0.5,
    smooth_idf=True)
X = vectorizer.fit_transform(documents)

svd_model = TruncatedSVD(n_components=20)
svd_model.fit(X)
```

Compression: $m \times n \rightarrow m \times k + n \times k + k \times k$

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors
- Jacobi rotation [4, 5], Jacobi eigenvalue algorithm [7]:
  Create almost a diagonal matrix (bidiagonal): $A = UBV$, $O(mn^2)$
  Compute SVD of 2×2 matricis $O(n^2)$

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors
- Jacobi rotation [4, 5], Jacobi eigenvalue algorithm [7]:
  Create almost a diagonal matrix (bidiagonal): $A = UBV$, $O(mn^2)$
  Compute SVD of 2×2 matricis $O(n^2)$
- Can be parallelized (ARPACK)

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors
- Jacobi rotation [4, 5], Jacobi eigenvalue algorithm [7]:
  Create almost a diagonal matrix (bidiagonal): $A = UBV$, $O(mn^2)$
  Compute SVD of 2×2 matricis $O(n^2)$
- Can be parallelized (ARPACK)

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors
- Jacobi rotation [4, 5], Jacobi eigenvalue algorithm [7]:
  Create almost a diagonal matrix (bidiagonal): $A = UBV$, $O(mn^2)$
  Compute SVD of 2×2 matricis $O(n^2)$
- Can be parallelized (ARPACK)

Latent Semantic Analysis

- Also called LSI (Latent Semantic Indexing)

Notes

Fast SVD

- Naive approach $det(A - \lambda I) = 0$ solving n-th order polynomial (variable λ)
  Eigenvector Decomposition (EVD), get eigenvectors
- Jacobi rotation [4, 5], Jacobi eigenvalue algorithm [7]:
  Create almost a diagonal matrix (bidiagonal): $A = UBV$, $O(mn^2)$
  Compute SVD of 2×2 matricis $O(n^2)$
- Can be parallelized (ARPACK)

Latent Semantic Analysis

- Also called LSI (Latent Semantic Indexing)
- tf-idf is just a weighting scheme (tf, counts)

Considerations

Pros:

- Easy to implement
- Explainable terms
- Quite fast runtime

Considerations

Pros:

- Easy to implement
- Explainable terms
- Quite fast runtime

Cons:

- Only surface dependencies
- SVD is not updatable

Homework

TBD

Resources

1. Python code:
   https://medium.com/acing-ai/what-is-latent-semantic-analysis-lsa-4d3e2d18417a
2. Comprehensive tutorial for LSA+SVD: https://www.engr.uvic.ca/~seng474/svd.pdf
3. SVD example:
   http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm
4. Computation:
   https://en.wikipedia.org/wiki/Singular_value_decomposition#Calculating_the_SVD
5. Computation: https://www.cs.utexas.edu/users/inderjit/public_papers/HLA_SVD.pdf
6. Visualization: https://topicmodels.west.uni-koblenz.de/ckling/tmt/svd_ap.html
7. Computation: https://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm
8. Python code: https://www.analyticsvidhya.com/blog/2018/10/stepwise-guide-topic-modeling-latent-semantic-analysis/
9. Jelinek-Mercer: http://ctp.di.fct.unl.pt/~jmag/ir/slides/a05%20Language%20models.pdf