

# Information Retrieval + Q&A

## (SNLP Tutorial 12)

Vilém Zouhar, Awantee Deshpande, Julius Steuer

13th July, 15th July

# Evaluation 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  $\frac{|\text{retrieved} \cap \text{relevant}|}{|\text{retrieved}|}$

## How to cheat so that...

- precision is high?
- recall is high?

## {Precision, Recall}@ $k$ : Retrieve $k$ documents (top $k$ scoring)

- Recall@ $k$   $\frac{|\text{retrieved}@k \cap \text{relevant}|}{|\text{relevant}|}$
- Precision@ $k$   $\frac{|\text{retrieved}@k \cap \text{relevant}|}{|\text{retrieved}@k|} = \frac{|\text{retrieved}@k \cap \text{relevant}|}{k}$

# Evaluation metrics

- Average precision:  $AveP(q) = \frac{\sum_k P@k \times rel(k)}{|relevant|}$
- $rel(k) = \begin{cases} 1 & k\text{-th document relevant} \\ 0 & otherwise \end{cases}$
- Mean average precision  $MAP(Q) = \frac{\sum_{q \in Q} AveP(q)}{|Q|}$
- ▶  $Q$  can be a “test set”
- F-score  $2 \cdot \frac{P \cdot R}{P + R}$

## Evaluation metrics

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

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

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

## └ Evaluation metrics

Lots of others, accuracy, r-precision etc. Papers (as compared to MT) usually use a lot of different metrics.

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

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

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

# 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*)
- - Not always: query *Tokyo* vs. *Tokio*

Always depends on the task.

# 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)
  - A (Wolfgang - Goethe, Mephistopheles - devil)
  - C (unrelated context)
- Can these inferences be made automatically?

## 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, ...).
- 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

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

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

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

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

## Augmented TF

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

## └ Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency

TF-IDF

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

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

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

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

Augmented TF

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

- Probability that i-th term occurs k times in the document:  $p_{\lambda_i}(k) = e^{-\lambda_i} \frac{\lambda_i^k}{k!}$  ( $\lambda_i$  parameter of the distribution)
- Expected value of occurrence:  $N \cdot E_i(k) = N \cdot \lambda_i = \text{collection frequency}_i$
- Term present at least once:  $N \cdot (1 - P_{\lambda_i}(0)) = \text{document frequency}_i$

## Solution 3 (tf-idf)

- Solution: vector of tf-idf
- Ranking: Cosine similarity between query and document vectors
- Good metrics to determine the significance of a term in a document collection
- Issue: still enormous vectors
- Issue: devil - Mephistopheles are equally separate concepts as devil - lasagne
- Issue: independent terms assumption

# Document Retrieval - Probabilistic Retrieval

- Goal: Find  $P[R|d,q]$
- Ranking: Proportional to relevance odds

$$O(R|d, q) = \frac{P[R|d, q]}{P[\bar{R}|d, q]}$$

- Different probabilistic models calculate these probabilities differently  
e.g. Binary Independence model, Poisson model, BM25

For Poisson,  $P[d|\lambda] = \prod_{t \in V} \frac{e^{-\lambda_t} \cdot \lambda_t^{d_t}}{d_t!}$

# Document Retrieval - Statistical Language Model

- Pretend the query was generated by a LM based on the document
- Ranking: Proportional to query likelihood
- $\operatorname{argmax}_d p(d|q) = \operatorname{argmax}_d \frac{p(q|d) \cdot p(d)}{p(q)} = \operatorname{argmax}_d p(q|d) \cdot p(d)$   
 $\approx \operatorname{argmax}_d p_{LM}(q|d) \cdot p(d)$   
 $p(d) \approx \frac{1}{|D|}$  or  $p(d)$  is query independent  
 $\approx \operatorname{argmax}_d p_{LM}(q|d)$
- Unigram:  $p(d|q) \approx \prod_i p_{LM}(q_i|d)$
- LMs can be smoothed, as you remember e.g. Interpolation, Dirichlet smoothing
- Jelinek-Mercer smoothing:  $p(q_i|d, D) = \lambda \cdot p(q_i|d) + (1 - \lambda) \cdot p(q_i|D)$   
High  $\lambda$ : documents with all query words (conjunctive)  
Low  $\lambda$ : 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
- Can we model word co-occurrence for a topic?

## └ Document Retrieval - Statistical Language Model

### Document Retrieval - Statistical Language Model

- ◆ Pretend the query was generated by a LM based on the document
- ◆ Ranking: Proportional to query likelihood
  - ◆  $\text{argmax}_d p(d|q) = \text{argmax}_d \frac{p(q|d)p(d)}{p(q)} = \text{argmax}_d p(q|d) \cdot p(d)$
  - $\approx \text{argmax}_d p_{LM}(q|d) \cdot p(d)$
  - $p(d) \approx \frac{1}{|D|}$  or  $p(d)$  is query independent
  - $\approx \text{argmax}_d p_{LM}(q|d)$
- ◆ Unigram:  $p(d|q) \approx \prod_i p_{LM}(q_i|d)$
- LMs can be smoothed, as you remember e.g. Interpolation, Dirichlet smoothing
- Jelinek-Mercer smoothing:  $p(q|d, D) = \lambda \cdot p(q|d) + (1 - \lambda) \cdot p(q|D)$ 
  - High  $\lambda$ : documents with all query words (conjunctive)
  - Low  $\lambda$ : 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
- Can we model word co-occurrence for a topic?

- Other smoothing schemas exist, like discounting, adding epsilon or linear interpolation between multiple LMs, including zerogram
- Other improvements, such as special grammar, prior knowledge of the document (length), list of synonyms, etc

## Solution 4 (Latent Semantic Analysis)

- Assumption: Documents are composed of  $k$  latent topics.
- Solution: Perform dimensionality reduction  $\rightarrow$  eigenvalues, singular value decomposition
- $A_{i,j} = \# \text{occurrences of term } t_i \text{ in document } d_j$

	$d_1$	$d_2$	$d_3$	$d_4$
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
lasagne	0	0	1	0
German	0	1	0	1



## └ Solution 4 (Latent Semantic Analysis)

### Solution 4 (Latent Semantic Analysis)

- Assumption: Documents are composed of  $k$  latent topics.
- Solution: Perform dimensionality reduction  $\rightarrow$  eigenvalues, singular value decomposition
- $A_{i,j}$  = #occurrences of term  $t_i$  in document  $d_j$

	$d_1$	$d_2$	$d_3$	$d_4$
Wolfgang	1	1	0	0
Mephistopheles	1	0	0	0
Faust	0	1	0	0
Gothie	0	1	0	1
devil	0	1	1	0
demon	1	0	0	1
lasagne	0	0	1	0
German	0	1	0	1

The example uses counts, but for better representation of term importance in the document, one would use tf-idf.

## Approximation of A

	$d_1$	$d_2$	$d_3$	$d_4$
Wolfgang	1	1	0	0
Mephistopheles	1	<b>1</b>	0	<b>1</b>
Faust	<b>1</b>	1	0	0
Goethe	<b>1</b>	1	0	<b>0</b>
devil	<b>1</b>	1	<b>0</b>	<b>1</b>
demon	1	<b>1</b>	0	1
lasagne	0	0	1	0
German	<b>1</b>	1	0	<b>0</b>

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

3 latent concepts:

{Goethe (Wolfgang, Faust, German), devil (Mephistopheles, demon), lasagne}

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

## Information Retrieval + Q&amp;A

└ Approximation of  $A$ 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
lasagne	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
lasagne	0	0	1
German	1	0	0

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

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

Given  $k$  concepts, we try to find such a matrix  $A'$ , that's as close to the original one, but with every document being a combination of  $k$  independent vectors.

# Approximation of $A$

- Given:  $A, k$
- $A' = \operatorname{argmin}_{A' \text{ rank } k} \|A - A'\|$
- Distance e.g. Frobenius ( $\sqrt{\sum_{i,j} a_{i,j}^2}$ )

# Singular Value Decomposition

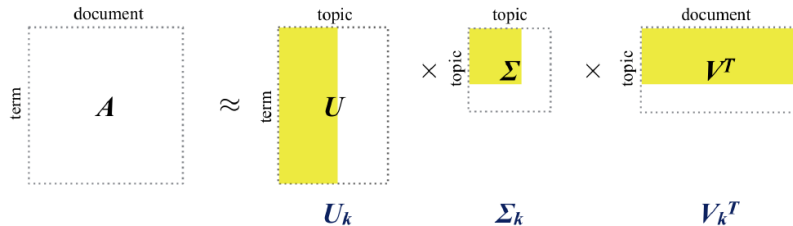


Figure 1: SVD for LSA

- $U$  = eigenvectors of  $A^T A$  (# intersection of documents  $d_i$  and  $d_j$ )
- $V$  = eigenvectors of  $AA^T$  (# documents in which both terms  $t_i$  and  $t_j$  occur)
- $S$  = 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

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

"Directions ( $v$ ) which  $A$  only scales."

## Eigenvalue

$$Av = \lambda v$$

"The stretch ( $\lambda$ ) of eigenvector  $v$  by  $A$ ."

->

## Proof sketch

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

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

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

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

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

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

- 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:  $(S_k V_k^T)^T = V_k S_k^T = V_k S_k$



## └ LSA

- Order eigenvalues by descending values ( $S_{i,i} > S_{i+1,i+1} \geq 0$ )  
(proof next slide)
- Take top-k eigenvectors + values (or all above threshold)
- $A_k = U_k S_k V_k^T$   $[(m \times n), (n \times n)] \rightarrow [(m \times k), (k \times k), (k \times n)]$ 
  - Term  $\rightarrow$  latent representation:  $U_k S_k$
  - Document  $\rightarrow$  latent representation:  $(S_k V_k^T)^T = V_k S_k^T = V_k S_k$

- We are free to permute the eigenvalues, so we can order them (together with the vectors) and also we know that the eigenvalues are non-negative
- Therefore we can just take the top-k eigenvalues and replace the rest with zero.
- Essentially this crops the neighbouring matrices to first k columns and first k rows of  $V^T$ .

# Properties of S

## Descending

$U' = U$  +swapped  $i, j$  column,  $S' = S$  +swapped  $i, j$  values,  $V'^T = V^T$  +swapped  $i, j$  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

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

$$\forall x \neq \vec{0} : x^T A^T A x = (Ax)^T (Ax) = \|Ax\|^2 \geq 0$$

# 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 Concepts

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

- The  $k$  then becomes obvious is the number of concepts
- We don't specify the concepts, they are determined by SVD
- From our point of view, they are latent

## 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  $S_k$ :  $[0.58, -0.01]^T$
- Representation of  $d_1$ : first column of  $V_k^T$  ( $k \times n \rightarrow 2 \times 1$ ) scaled first by  $S_k$ :  
 $r_d = [0.3, 0.02]^T$
- Map query to our topic space:  $q \rightarrow U_k^t \cdot q = q' = [0.355, -0.07]^T$
- Query-document match: dot product, cosine similarity:  $\frac{r_q \cdot r_d}{|r_q| \cdot |r_d|} = \frac{0.01205}{0.10879} \approx 0.11$

## └ LSA Example

## 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  $S_k$ :  $[0.58, -0.01]^T$
- Representation of  $d_1$ : first column of  $V_k^T$  ( $k \times n \rightarrow 2 \times 1$ ) scaled first by  $S_k$ :  
 $r_d = [0.3, 0.02]^T$
- Map query to our topic space:  $q \rightarrow U_k^T \cdot q = q' = [0.355, -0.07]^T$
- Query-document match: dot product, cosine similarity:  $\frac{q' \cdot r_d}{\|q'\| \|r_d\|} = \frac{0.01266}{0.18879} \approx 0.11$

- Whether that's a good match or not depends on the ranking and/or threshold

# LSA Graphics

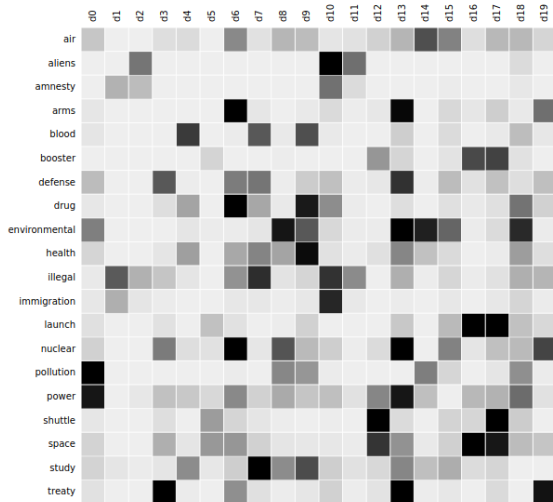


Figure 2: Term-document matrix, no ordering,  $k = 5$ ; Source [6]

# LSA Graphics

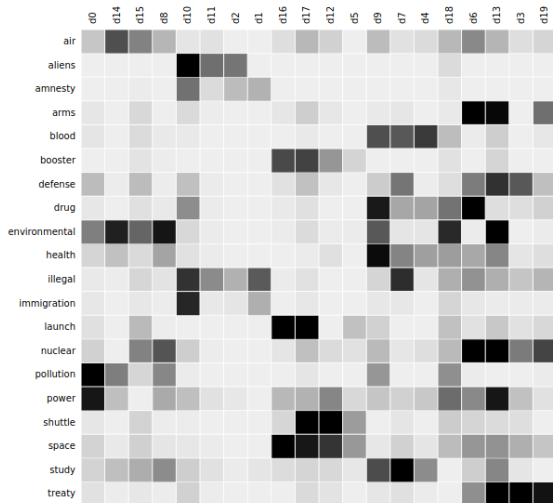


Figure 3: Term-document matrix, group documents,  $k = 5$ ; Source [6]



# LSA Graphics

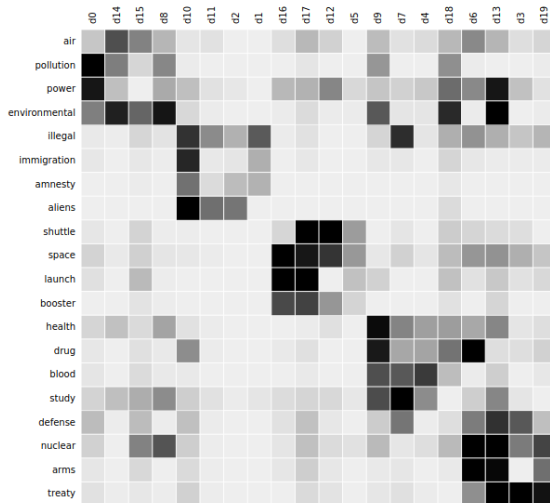


Figure 4: Term-document matrix, group documents+terms,  $k = 5$ ; Source [6]

## 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)
```

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

## └ LSA Code

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

- max\_features takes to top 1000 terms, max\_df removes all words which appear in at least half the documents.
- smooth\_idf adds one to ever seen term
- The reason it's called Truncated SVD is because it can be used for matrix compression. Instead of transmitting  $m \times n$  matrix, we can just transmit the three separate matrices.

# Considerations

## Notes:

- tf-idf is just a weighting scheme (tf, counts)
- SVD naive approach  $\det(A - \lambda I) = 0$  solving  $n$ -th order polynomial (variable  $\lambda$ )  
Eigenvector Decomposition (EVD), get eigenvectors
- Faster, approximate methods available

## Pros:

- Easy to implement
- Explainable terms
- Quite fast runtime
- Handles synonymy of words

## Cons:

- Only surface dependencies
- Determination of  $k$
- SVD difficult to update

# Dense Vectors

- (Sentence)BERT (CLS):  $D \cup Q \rightarrow \mathbb{R}^{768}$
- $h_q = \text{BERT}(\textit{Goethe devil})$
- $h_a = \text{BERT}(\textit{Wolfgang's idea of the demon Mephistopheles who makes a bet with God})$
- $h_c = \text{BERT}(\textit{Devilishly good lasagne})$
- $h_q \cdot h_a = 14.1, h_q \cdot h_c = 0.9$
- Used in industry (with better models than BERT)

# 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](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](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](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](https://topicmodels.west.uni-koblenz.de/ckling/tmt/svd_ap.html)
- ⑦ Computation: [https://en.wikipedia.org/wiki/Jacobi\\_eigenvalue\\_algorithm](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>
- ⑩ LSI: <https://nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html>