

universität freiburg

Databases and Information Systems WS 25/26

Lecture 11: Vector Space Model, Word Embeddings
January 20, 2026

Prof. Dr. Hannah Bast
Department of Computer Science
University of Freiburg

Overview of this lecture

■ Organizational

- Your experiences with ES10
- Demo of some web apps
- Official course evaluation

Web apps, part 2

drumroll

Please fill it out asap

■ Contents

- Vector Space Model (VSM) documents as vectors
- Word embeddings words as vectors
- ES11: refactor your search engine from ES2 using the vector-space model + implement a simple similarity search based on word embeddings we provide for you

Experiences with ES10 1/2

■ Excerpts from your feedback [only few submissions]

"Again very interesting. I'm actually really enjoying the topic and would have liked to spend a lot of time learning this stuff"

"Lot of fun and I am looking forward to finishing this up"

"The easter eggs were a pretty good idea to teach people what code injection can really do with their machines"

matrix, gorillas, harlem, snow, asteroids

■ Who is the president of Venezuela?

"Not the orange head!"

"I believe that only a new and fair democratic election should answer that question"

"Can I be sure nobody from the US is reading this answer?"

Experiences with ES10 2/2

■ Will this be on the exam?

- Yes, there is always a question in the exam that involves writing some server code and/or HTML and/or JavaScript and requires understanding how this works in principle

If you have done the exercise sheets that will be easy

We don't see how you can learn this stuff without doing the exercise sheets or something equivalent

We always have participants in our exams who make the impression of having never written a web app themselves, and then don't get the points ... **don't be one of them**

Official course evaluation

■ Instructions

- You should have received an email from EvaSys Admin on Monday, January 12 with a link to an evaluation form

Please take your time for this and be honest, concrete, fair *

The free text comments are particularly interesting for us

Your feedback is very important and interesting for us,
please complete until Sunday, January 25 the very latest

* One of the questions asks how much (or not) you worked on the **exercise sheets** ... please provide a truthful answer (it's anonymous)

- The evaluation is centralized, and once it is closed, there is nothing we can do about that anymore
- If you did not receive any email or if you have any other question, please let us know in the subforum "Evaluation"

Vector Space Model 1/11

■ Motivation

*internet - web : "synonym"
surfing : "polysem"*

- For this lecture, it will be useful to represent documents as **vectors** ... here is our running example for today:

	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

- Each row corresponds to a word, each column to a document
- Non-zero entries: score for that word in that document

In the lecture, we use **tf** scores ... for ES11, use **BM25** scores

Vector Space Model 2/11

■ Terminology

- Often referred to as the **Vector Space Model (VSM)**
- In the VSM, words are traditionally referred to as **terms**
- Putting the vectors from all documents from a given corpus side by side gives us the so-called **term-document matrix**

	D₁	D₂	D₃	D₄	D₅	D₆
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Vector Space Model 3/11

■ Retrieval

$$x = (x_1, \dots, x_m)$$
$$y = (y_1, \dots, y_m)$$
$$x \bullet y = \sum_{i=1}^m x_i \cdot y_i$$

- A query can also be represented as a vector ... we take **1** for a term used in the query, and **0** for all other terms
- We measure the relevance of a document to the query by taking the **dot product** of the two vectors

Note: corresponds to the **tf** scores from Lecture 2, Slide 10

	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	Q
internet	1	1	0	1	0	0	0
web	1	0	1	1	0	0	1
surfing	1	1	1	2	1	1	1
beach	0	0	0	1	1	1	0
	2	1	2	3	1	1	

Vector Space Model 4/11

■ Linear Algebra

$$\underbrace{\begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}}_{q^T \ 1 \times 4} \underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}}_{A \ 4 \times 6} = \underbrace{\begin{bmatrix} 2 & 1 & 2 & 3 & 1 & 1 \end{bmatrix}}_{q^T \cdot A \ 1 \times 6}$$

- More formally, let us write A for the term-document matrix and q for the query vector
- Then the matrix-vector product $q^T \cdot A$ gives us a vector with the relevance scores of all the documents

Let us implement this together now

		D_1	D_2	D_3	D_4	D_5	D_6	Q
0	internet	1	1	0	1	0	0	0
1	web	1	0	1	1	0	0	1
2	surfing	1	1	1	2	1	1	1
3	beach	0	0	0	1	1	1	0

4×6 4×1

Vector Space Model 5/11

■ Basic linear algebra in Python

- For this and the next lecture, we use **PyTorch**

In previous years, we used the older **numpy** and **scipy**

- There is ample documentation on the web, we have provided some links on our Wiki as well as a [PyTorch cheat sheet](#)
- Here is some example code + see code from this lecture

```
import torch
```

```
A = torch.tensor([[1, 1, 0, 1, 0, 0], ..., [0, 0, 0, 1, 1, 1]])
```

```
q = torch.tensor([[0, 1, 1, 0]])
```

```
scores = q @ A
```

Note: In PyTorch vectors and matrices are both "tensors", a vector is a 1D-tensor and a matrix is a 2D-tensor

■ Document similarity, variant 1

- **Problem:** when we take the dot product to compute scores, documents with more and/or higher entries get larger scores
- One way to alleviate this is to take scores like BM25

Recall that BM25 was explicitly designed to handle different document lengths and different term frequencies such that scores reflect relevance in the best possible way

And understand that with BM25 scores, the dot product gives **exactly** the same ranking as in Lecture 2

- For Exercise 1 of ES11, take dot product with BM25 scores

Vector Space Model 7/11

■ Document similarity, variant 2

- Compute the similarity between two vectors as follows:

$$\text{cos-sim}(x, y) := x \bullet y / (|x| \cdot |y|)$$

$$= \cos \alpha = \cos \angle(x, y)$$

where $\bullet$ is the dot product between two vectors and

$|\dots|$ or $|\dots|_2$ is the L_2 -norm of a vector, defined as

$$|(x_1, \dots, x_n)| := \sqrt{x_1^2 + \dots + x_n^2}$$

- This similarity measure is called **cosine similarity** because it is equal to the cosine of the angle between the two vectors

Law of cosines: $c^2 = a^2 + b^2 - 2ab \cos \angle(a, b)$

$$= 0 \text{ if } \angle(a, b) = 90^\circ$$

PROOF: [nice exercise]

$$x \bullet y / (|x| \cdot |y|) = \cos \angle(x, y)$$

- For Exercise 2 of ES11, use the cosine similarity

$$\begin{aligned} x &= (x_1, \dots, x_n) \\ y &= (y_1, \dots, y_n) \\ x \bullet y &= \sum_{i=1}^n x_i y_i \\ |x| &:= \sqrt{\sum_{i=1}^n x_i^2} \\ &= \sqrt{x \bullet x} \end{aligned}$$



$$\begin{aligned} \angle(a, b) &= 90^\circ \\ \Rightarrow \cos \angle(a, b) &= 0 \\ \Rightarrow \text{Pythagorean THEOREM} \end{aligned}$$

Vector Space Model 8/11

■ Efficient computation of the cosine similarity

- For Exercise 2 of ES 11 you need to compute the cos-sim between each document and the given query repeatedly

Computing n dot products can be done efficiently with a single matrix-vector multiplication ... see slides 8 and 9

But computing $n + 1$ times the L_2 -norm of a vector is costly

- Instead, you can normalize the vectors **once** in advance and afterwards get away with only computing dot products

Let $x' = x / |x|$ and $y' = y / |y|$

Then $|x'| = |y'| = 1$ and $\cos(x, y) = \cos(x', y') = x' \bullet y'$

$$|x'| = \left| \frac{x}{|x|} \right| = \frac{1}{|x|} \cdot |x| = 1$$

$$|y'| = \dots = 1$$

$$x' \bullet y' = \frac{x}{|x|} \bullet \frac{y}{|y|}$$

$$= \frac{x \bullet y}{|x| \cdot |y|} = \cos \angle(x, y)$$

Vector Space Model 9/11

■ Sparse matrices

#terms = 1M
#docs = 10M $\Rightarrow$ size of A
 $= 1M \cdot 10M$
 $= 10^7$

- Storing **all** entries of a large term-document matrix is infeasible

But most entries of a typical term-document matrix are **zero**

- Sparse-matrix representation: store only the non-zero entries, together with their row and column index

$(1, 0, 0), (1, 0, 1), (1, 0, 3), \dots, (2, 2, 3), \dots$

		0	1	2	3	4	5
		D ₁	D ₂	D ₃	D ₄	D ₅	D ₆
0	internet	1	1	0	1	0	0
1	web	1	0	1	1	0	0
→ 2	surfing	1	1	1	2	1	1
3	beach	0	0	0	1	1	1

Vector Space Model 10/11

■ Sparse matrices

- Two principle ways to store the list of non-zero values

row-major: store row by row (sort by row index first)

column-major: store col by col (sort by col index first)

- Note: the sparse row-major representation of a term-document matrix is equivalent to an **inverted index**

(1, 0, 0), (1, 0, 1), (1, 0, 3) ids of docs containing term 0

(1, 1, 0), (1, 1, 2), (1, 1, 3) ids of docs containing term 1

(1, 2, 0), (1, 2, 1), (1, 2, 2) ... ids of docs containing term 2

(1, 3, 3), (1, 3, 4), (1, 3, 5) ids of docs containing term 3

(non-zero score, row index = term id, col index = doc id)

Vector Space Model 11/11

■ Sparse matrices in PyTorch

- To efficiently initialize a sparse matrix in PyTorch, specify the non-zero values and the corresponding row and col indices

```
values    = [1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1]
rows      = [0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3]
cols      = [0, 1, 3, 0, 2, 3, 0, 1, 2, 3, 4, 5, 3, 4, 5]
A_sparse = torch.sparse_coo_tensor(
    torch.tensor([rows, cols]), values, dtype=torch.int)
```

Note: the "coo" simply stands for "coordinates" ... internally, the non-zero values are always stored in row-major order

- In PyTorch, you can multiply a sparse with a dense tensor

```
q = torch.tensor([[0, 1, 1, 0]], dtype=torch.int)
scores = q @ A_sparse
```

Embeddings 1/6

■ Vocabulary

- We call the set of distinct words of a document collection its **vocabulary**, often denoted by V ... for our running example:

$V = \{\text{"internet"}, \text{"web"}, \text{"surfing"}, \text{"beach"}\}$

- If we write V as an **ordered set** (that is, a sequence), each word corresponds to an integer ... the index in that sequence

$V = (\text{"internet"}, \text{"web"}, \text{"surfing"}, \text{"beach"})$

0 : internet

1 : web

2 : surfing

3 : beach

The order typically does not matter, as long as we stick to it

■ One-hot encoding

- Given an ordered vocabulary V , we can write each word from the vocabulary as a vector as follows

$[0, 0, \dots, 0, 1, 0, \dots, 0]$

where the dimension of the vector is $|V|$ and **the value at the index of the word in V is 1**, all other values are 0, e.g.

internet = $[1, 0, 0, 0]$

web = $[0, 1, 0, 0]$

beach = $[0, 0, 1, 0]$

surfing = $[0, 0, 0, 1]$

- This representation is called **one-hot encoding**

Note that this can be done for any ordered set (not just words)

■ Relation of one-hot encoding to vector space model

- In a term-document matrix, each document is simply the **sum** of the one-hot encodings of the words it contains

document = "internet web surfing beach surfing"

internet = [1, 0, 0, 0] $\times 1$

web = [0, 1, 0, 0] $\times 1$

surfing = [0, 0, 1, 0]

beach = [0, 0, 0, 1] $\times 1$

surfing = [0, 0, 1, 0] $\times 2$

document = [1, 1, 2, 1]

- Note that the same is true for the query vector

Since a query typically contains few keywords and each at most once, the query vector typically has a few 1s, and 0s otherwise

■ Properties of one-hot encoding

- In a one-hot encoding, the representations of each pair of elements that are not the same are **orthogonal** to each other

That is, their dot-product (and hence their cos-sim) is zero, e.g.

internet = [1, 0, 0, 0]

web = [0, 1, 0, 0]

internet = [1, 0, 0, 0]

beach = [0, 0, 1, 0]

Wouldn't it be nice if words that are semantically similar (like "internet" and "web") have a higher cos-sim than words that are semantically dissimilar (like "internet" and "beach") ?

■ Word embeddings

- Also a vector for each element from a vocabulary V , with the following differences to the one-hot-encoding

1. The dimension d of the vector does **not** have to be V

Typically $d \ll |V|$... for ES11, $d = 300$ and $|V| = 129\,603$

2. The vectors are **not** pairwise orthogonal

Good embeddings have the property that vectors of semantically related words are similar (e.g., regarding cos-sim), and vectors of semantically unrelated words are not

- For ES11, we provide such vectors for you

Unfortunately, we don't have time in this course to explain how such vectors are computed ... but see [IR WS 22/23, Lecture 9](#)

■ Using the embeddings provided for ES11

- The embeddings we provide are simply a **dictionary**, where each key is a word and the value of a key is the embedding of the respective word ... for example:

```
embs = torch.load("embeddings.pt")
emb_internet = embs["internet"]
emb_web = embs["web"]
sim = torch.cosine_similarity(emb_internet, emb_web, dim=0)
```

Note: the `dim=0` is required in `PyTorch`, even when the arguments are one-dimensional tensors (that is, vectors)

- To get the embedding of a document, simply take the sum of the embeddings of the contained words (like on slide 19), e.g.

```
doc_emb = embs["internet"] + embs["web"] + embs["surfing"]
```

References

■ Textbook

Section 6.3: The vector space model for scoring

■ Linear algebra in PyTorch ... <https://pytorch.org>

- Basic usage of tensors:

<https://pytorch.org/tutorials/beginner/ptcheat.html#tensors>

- Cheat sheet from us

<https://ad-wiki.informatik.uni-freiburg.de/teaching/PyTorchCheatSheet>

- PyTorch for numpy users

<https://github.com/wkentaro/pytorch-for-numpy-users>