https://miro.com/app/board/o9J_lNQKQbw=/

Index Compression + Vector Space Model (Chapter 5+6)

Exercise 6/1

Consider the frequency table of the words of three documents.
Calculate the **tf-idf weight** of the terms **car**, **auto**, **insurance**, and **best** for each document.
idf values of terms are in the table.

	doc_1	doc_2	doc_3	idf
car	27	4	24	1.65
auto	3	33	0	2.08
insurance	0	33	29	1.62
best	14	0	17	1.5

term frequency × idf

	tf-idf		
	doc_1	doc_2	doc_3
car	44.55	6.6	39.6
auto	6.24	68.64	0
insurance	0	53.46	46.98
best	21	0	25.5

Definition 6 (Inverse document frequency)
Inverse document frequency of a term t is defined as

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

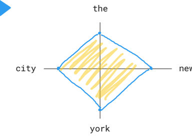
where N is the number of all documents and df_t (the document frequency of t) is the number of documents that contain t.

Definition 7 (tf-idf weighting scheme)
In the tf-idf *weighting scheme, a term t in a document d has weight*

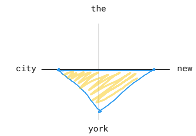
$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t$$

where $tf_{t,d}$ is the number of tokens t (the term frequency of t) in a document d.

"the new york city"

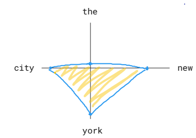


"new york city"



idf = log (#documents / #documents with 'the')
~ log (1)
~ 0

"the new york city"



Exercise 6/2

Count document representations as normalized Euclidean weight vectors for each document from the previous exercise. Each vector has four components, one for each term.

Represent the documents and the query as vectors:

```
doc_1 = [1,1,0,1]
doc_2 = [1,1,1,0]
doc_3 = [1,0,1,1]

q = 'car insurance'
q = [1,0,1,0]
```

Different approaches / views:

a) Boolean retr. results: {doc1, doc2, doc3} } OR operator

b) Ranked retr. results: query * doc } dot / scalar product

```
doc_1 = 1*1+0*1+1*0+0*1 = 1
doc_2 = 1+1 = 2
doc_3 = 1+1 = 2
```

Exercise 6/3

Based on the weights from the last exercise, compute the **similarity scores** (scalar products) of the three documents for the query **Q: "car insurance"**.

Use each of the two weighting schemes:
a) Term weight is 1 if the query contains the word and 0 otherwise.
b) Euclidean normalized **tf-idf**.

doc_2 = [4, 33, 33, 0]
doc_3 = [24, 0, 29, 17]

c) using TF (term frequency):

```
doc_2 = 4 + 33 = 37
doc_3 = 24 + 29 = 53
```

BEST HIT ?

d) using IDF (inverse document frequency):
TF-IDF + Euclidean normalization

Exercise 6/4

Compute the Levenshtein distance between paris and alice.

Write down the matrix of distances between all prefixes as computed by Algorithm 2.

	ε	p	a	r	i	s
ε	0	1	2	3	4	5
a	1					
l	2					
i	3					
c	4					
e	5					

Table 4: Initialization of the matrix.

movement: → / ↓ / ↘ — operation (price):
- delete (1)
- insert (1)
- replace (1) / move (0)

	ε	p	a	r	i	s
ε	0	1	2	3	4	5
a	1	1	1			
l	2					
i	3					
c	4					
e	5					

Table 5: First two iterations of the main dynamic programming step.

Algorithm 2 (Levenshtein distance – imperative approach)

```
function LevenshteinDistance(s1, s2)
    for i = 0 to |s1| do
        m[i,0] = i
    end for
    for j = 0 to |s2| do
        m[0,j] = j
    end for
    for i = 1 to |s1| do
        for j = 1 to |s2| do
            if s1[i] == s2[j] then
                m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]}
            else
                m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]+1}
            end if
        end for
    end for
    return m[|s1|,|s2|]
end function
```

Algorithm 3 (Levenshtein Distance – declarative approach)
Distance between two strings a and b is given by $lev_{a,b}(|a|,|b|)$ where

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 1 when $a_i \neq b_j$, and 0 otherwise. $lev_{a,b}(i,j)$ is the distance between the first i characters of string a and the first j characters of string b.

	ε	p	a	r	i	s
ε	0	1	2	3	4	5
a	1	1	1	2	3	4
l	2	2	2	2	3	4
i	3	3	3	3	2	3
c	4	4	4	4	3	3
e	5	5	5	5	4	4

Table 6: The final matrix with the Levenshtein distance in bold.

(yellow)		(red)	
paris		paris	
aris (1)	→	aris (1)	→
aris (0)	↘	aris (0)	↘
alis (1)	↘	alis (1)	↘
alis (0)	↘	alis (0)	↘
alic (1)	↘	alics (1)	↓
alice (1)	↓	alice (1)	↘

Exercise 5/5

From the following sequence of **γ-encoded** gaps, reconstruct first the gaps list and then the original postings list. Recall that the α code encodes a number n with n 1s followed by one 0.

1110001110101011111101101111011

1001	110	11	11011	111	(binary) } gaps
9	6	3	59	7	(decimal) } gaps

[9, 15, 18, 77, 84] (posting list)

Definition 1 (α code)
Unary code, also referred to as α code, is a coding type where a number n is represented by a sequence of n 1s (or 0s) and terminated with one 0 (or 1). That is, 6 in unary code is 1111110 (or 0000001). The alternative representation in parentheses is equivalent but for this course we use the default representation.

Definition 2 (γ code)
γ code is a coding type, that consists of an offset and its length: $\gamma(n) = \alpha(\text{length of offset}(n)), \text{offset}(n)$. Offset is a binary representation of a number n without the highest bit (1). The length of this offset encoded in the unary (α) code. Then the number 60 is encoded in γ as 111110,11100.

Q: how to encode '1' in gamma code?
- 1 (decimal / unary)
- ↓
- ∅ (offset of length 0)
- ↓
- 0 (zero; α-code of the offset)
- ↓
- 0 (zero; γ-code of '1')

Example

0110111100 (γ-code)

1	111	10	(binary) (add leading '1' to the offset)
1	7	2	(decimal)

[1, 8, 10] (posting list)