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

	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

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

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

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.

→ NORM
→ NORMED TF-IDF |DOC 1| =

$$\sqrt{6{,}6^2 + 68{,}64^2 + 53{,}46^2 + 0^2}$$

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"**.

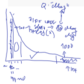
Use each of the two weighting schemes:
a) Term weight is 1 if the query contains the word and 0 otherwise.
b) Euclidean normalized **tf-idf**.

	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

Exercise 6/4

Compute the Levenshtein distance between paris and alive.

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					
v	4					
e	5					

Table 4: Initialization of the matrix.

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

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

Algorithm 2 (Levenshtein distance – imperative approach)

```
1:  function LEVENSHTEINDISTANCE(s1, s2)
2:      for i = 0 to |s1| do
3:          m[i, 0] = i
4:      end for
5:      for j = 0 to |s2| do
6:          m[0, j] = j
7:      end for
8:      for i = 1 to |s1| do
9:          for j = 1 to |s2| do
10:             if s1[i] == s2[j] then
11:                 m[i, j] = min{m[i-1, j] + 1, m[i, j-1] + 1, m[i-1, j-1]}
12:             else
13:                 m[i, j] = min{m[i-1, j] + 1, m[i, j-1] + 1, m[i-1, j-1] + 1}
14:             end if
15:         end for
16:     end for
17:     return m[|s1|, |s2|]
18: end function
```

	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
v	4	4	4	4	3	3
e	5	5	5	5	4	**4**

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

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.