Task 1: Vector Space Retrieval (theoretical)

In the script, we have used the inner vector product and the cosine measure to sort documents by their similarity to the query. In this task, we study the “semantics” of these functions from a geometrical perspective. To simplify matters, consider a query with only one term and two terms, and then generalize to higher dimensions.

a) Consider first a query with two terms and define a similarity threshold α. For both measures, identify the sub-space of documents that have a similarity score beyond α. Describe the space in geometrical terms.

b) Based on the geometrical semantics from a), identify the documents that are preferred by the measures. Construct an example document that “wins” the search (has highest scores). Generalize to queries with more than two terms.

c) In web search, queries are often very short. What happens if you only select one query term? Are the measures working in this extreme case?

We want to perform similarity search for texts (e.g., find pages that have stolen my content). We can use the bag-of-words model and compare the two texts by a Euclidean distance measure. Assume that $\boldsymbol{q}$ denotes the term vector for the Query Q, and $\boldsymbol{d}$ is the term vector of a document D. Then:

$$\delta(Q,D) = \sqrt{\sum_i (q_i - d_i)^2}$$

In contrast to the inner vector product and the cosine measure, small distances are better (more relevant) than large distances (less relevant).

d) Similar to a), describe the sub space of documents that have at most a distance of β to the query Q. What documents rank highest with this distance measure? Does this work in our scenario (finding similar pages) and why?

Task 2: Probabilistic Retrieval (theoretical)

In this task, we study the **binary independence retrieval (BIR)** model and use simple examples to run through the approach.

a) For a query Q, the BIR method yields the following list of documents after the initialization step:

rank	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
x_1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
x_2	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0
relevance	R	R	R	R	N	R	R	R	R	N	N	R	R	R	N	N	N	R	N	N

In the table above, the row x_1 and x_2 contain the binary representation of the 20 retrieved documents. The last row denotes the relevance assessment of the user for each document (R denotes relevant, N denotes non-relevant). Compute the new c_j-values given the feedback and compute the ordering.

b) The BIR model makes three assumptions. We now test whether these assumptions hold true. To this end, we compute the probability $P(R|\boldsymbol{x})$ with the example data from a) in two ways: 1) count how often a document with representation $\boldsymbol{x}$ is relevant/non-relevant and compute the probability. 2) derive a formula for $P(R|\boldsymbol{x})$ depending on r_j and n_j similarly to the script. Start with the following statement

$$sim(Q, D_i) = \frac{P(R|D_i)}{P(NR|D_i)} = \frac{P(R|D_i)}{1 - P(R|D_i)} = \frac{P(R|\boldsymbol{x})}{1 - P(R|\boldsymbol{x})} = \cdots$$

and solve for $P(R|\boldsymbol{x})$. What do you observe? Which assumption fails?

c) Consider the documents below (c1-c5, m1-m4) and the query **“human computer interaction”**. Conduct two iterations with the BIR model (initialization step, one feedback step) and assume that documents c1-c5 are relevant and m1-m4 are non-relevant. Does the feedback step help? What can we do to significantly improve retrieval performance with the feedback?

c1	**Human** machine interface for Lab ABC **computer** applications
c2	A survey of user opinion of **computer** system response time
c3	The EPS user interface management system
c4	System and **human** system engineering testing of EPS
c5	Relation of user-perceived response time to error measurement
m1	The generation of random, binary, unordered trees
m2	The intersection graph of paths in trees
m3	Graph minors IV: Widths of trees and well-quasi-ordering
m4	Graph minors: A survey

Task 3: Searching with Lucene (practical)

In this exercise, we use Lucene and its fuzzy retrieval model to search for music files. The web site of the course contains a list of file names, but you can also use your own music library.

- Download Lucene from Apache. Choose the programming language that fits you the best.
- Write a program to read the MP3 file names, create the index, and search for the titles that match your query. You can also use RAMDirectory for a fast implementation (but you need to build the index every time again)
- Extend the basic search with an implementation of the "Did you mean?" function that Google provides. If the query contains spelling mistakes (or is seldom), automatically search with the closest matches of the terms used.
  - Hint: Consider using the SpellChecker of Lucene