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: NLTK and Python (practical)

In this task, we use the NLTK library for Python to run a number of interesting text analytics. The next page contains a few hints how to setup NLTK (takes at most 5 minutes) and how to get started. You can either create your own classes and methods in Python, or simply collect the commands in a text file and copy paste to the interpreter.

a) **[easy]** Use NLTK to guess the language of an input text. Download an Italian, German, and English (or any other Language, preferably all in the same encoding to simplify matters). Use the stop word lists in NLTK to identify the language of the text. Try some harder text examples with mixed languages and return probabilities for the languages.
Hint: limit your analysis to a few fixed languages only

b) **[intermediate]** Assume we are using a service like https://www.clarifai.com to annotate images. Given a picture, we obtain a list of trained keywords associated with that picture. In order to broaden the keyword list, we want to extend each term with a set of related terms using WordNet. Use the online version of WordNet (http://wordnetweb.princeton.edu/perl/webwn) to get an idea how to do this cleverly. Then implement your idea with the NLTK corpus `nltk.corpus.wordnet`. See online documentation: Chapter 5 in http://www.nltk.org/book/ch02.html

c) **[difficult]** When translating from one language to another, a common problem is the wrong usage of words in the target language due to overlapping word semantics. For instance, the German word “stark” can have a number of English counterparts, namely: strong, intense, powerful, massive, potent, robust, vigorous, severe, heavy, thick, deep, and so on. Obviously, not all English counterparts are correct in a given context. Consider the following example:

- es regnet stark → it is raining hard / heavily (maybe: intensely / thickly)
  But not: it is raining strongly / powerfully / deeply / robustly / hardly
- der Mann ist stark → the man is strong / powerful
  But not: the man is robust / potent / heavy / thick / deep / hard
  (some combinations are possible but have different meaning)

n-grams (within windows) provide a simple way to identify the right word combinations (and also the right inflection, e.g., is it thick or thickly?). If we analyze an entire corpora of English books, we may find that the combination “rain, powerful” is less frequent than “rain, hard”. From that, we may infer the right word in the context. To simulate that process, write a Python script that takes the beginning of a sentence and completes it with the most frequent n-grams it finds in an example text (e.g., the Sherlock Holmes book referred to below). Use 3-grams and 4-grams, and match all but the last terms with the end of the sentence and extend the sentence with the most frequent matching n-gram. Repeat until you run into a punctuation (don’t eliminate punctuations). Look at the results!

Task 3: NLTK and Python (practical)

Getting started with NLTK (see also: http://www.nltk.org/install.html)

1. Install Python (https://www.python.org)
   - Ubuntu: `sudo apt-get install -y python3-pip python3-dev`
   - Windows: install python version (including pip)
2. Install Python packages with `pip` (or `pip3`) – use PowerShell on Windows
   - `pip install --upgrade pip`
   - `pip install -U numpy`
   - `pip install -U nltk`
   - `python -m nltk.downloader all` (alternative data download below)
3. Run `python` (or `python3`) – use PowerShell on Windows
   - `import nltk`
   - Alternative data download: `nltk.download()` [select all in dialog]
   - …write your commands (see below)

References:

- NLTK: http://www.nltk.org
- NLTK Book: http://www.nltk.org/book/
  - best source to find snippets of Python code for NLTK
- Python: https://www.python.org/doc/

How to get started with the exercise: (don't forget to `import nltk`)

- Read file from local folder (e.g., http://www.gutenberg.org/files/244/244-0.txt)
  - `f=open('stud.txt')`
  - `text=f.read()`
  - `f.close()`
- A few lines from the demo during the course:

```
import nltk
sentences=nltk.sent_tokenize(text)
tokens=nltk.word_tokenize(text)
words=[word.lower() for word in tokens if word.isalpha()]
bigram_measures=nltk.collocations.BigramAssocMeasures()
finder=nltk.collocations.BigramCollocationFinder.from_words(words)
finder.nbest(bigram_measures.pmi, 20)
finder.score_ngrams(bigram_measures.pmi)
nltk.pos_tag(tokens)
nltk.FreqDist(tag for (word, tag) in nltk.pos_tag(tokens)).most_common()
porter=nltk.PorterStemmer()
porter.stem("house")
nltk.corpus.wordnet.synsets("dog")
nltk.corpus.stopwords.words("English")
nltk.corpus.stopwords.words("german")
nltk.corpus.stopwords.words("Italian")
```