LAB 1 – Information Representation

Indexation

Objective

The purpose of this lab is to introduce you to the fundamental steps of document representation and indexing in an Information Retrieval system. You will learn how to read and preprocess a collection of documents, extract and normalize terms, remove stopwords, and perform stemming. Another key goal is the creation of a descriptor file based on term weighting, preparing you for later stages such as similarity computation and document ranking.

By the end of this lab, you will understand how the raw text of documents is transformed into a structured representation suitable for information retrieval operations.

1. Libraries installation

Open the Anaconda Prompt, activate the (**ir_env**) and install the below list of Python libraries.

Library	Install Command
numpy	**conda install numpy**
pandas	**conda install pandas**
matplotlib	**conda install matplotlib**
regex	**pip install regex**
nltk	**conda install nltk**
scikit-learn	**conda install scikit-learn**

2. Automatic Term Extraction – Tokenization/Segmentation

We start with a simple example of text

```
Text = """In 2025, Dr. A.I. Research-Lab released GPT-5.0 with
1.75B parameters – costing $120.50M!It outperforms BERT, RoBERTa, etc...
and supports 100+ languages."""
```

1.1 Extraction with the split() method

```
Terms= Text.split()

print("Extracted terms:\n", Terms)
```

Output:

```
Extracted terms:
 ['In', '2025,', 'Dr.', 'A.I.', 'Research-Lab', 'released', 'GPT-5.0', 'with',
'1.75B', 'parameters', '—', 'costing', '$120.50M!It', 'outperforms', 'BERT,',
'RoBERTa,', 'etc...', 'and', 'supports', '100+', 'languages.']
```

1.2 Extraction using NLTK (Regular Expressions)

We can use NLTK's RegexpTokenizer to extract terms more accurately.

```
import nltk
from nltk.tokenize import RegexpTokenizer
```

Example 1 – Basic alphanumeric extraction

```
ExpReg = RegexpTokenizer(r'\w+')
Terms = ExpReg.tokenize(Text)
print(Terms)
```

Output :

```
['In', '2025', 'Dr', 'A', 'I', 'Research', 'Lab', 'released', 'GPT', '5', '0',
'with', '1', '75B', 'parameters', 'costing', '120', '50M', 'It', 'outperforms',
'BERT', 'RoBERTa', 'etc', 'and', 'supports', '100', 'languages']
```

Example 2 – Preserving abbreviations (e.g., A.I., Dr.)

```
ExpReg = RegexpTokenizer(r'(?:[A-Z]\.)+|\w+')
Terms = ExpReg.tokenize(Text)
print(Terms)
```

Output:

```
['In', '2025', 'Dr', 'A.I.', 'Research', 'Lab', 'released', 'GPT', '5', '0',
'with', '1', '75B', 'parameters', 'costing', '120', '50M', 'It', 'outperforms',
'BERT', 'RoBERTa', 'etc', 'and', 'supports', '100', 'languages']
```

Example 3 – Words, Decimal numbers, abbreviations, and ellipses.

```
ExpReg = RegexpTokenizer(r'(?:[A-Z]\.)+|\d+(?:\.\d+)?[A-Za-z]*|\w+|\.{3}')
Terms = ExpReg.tokenize(Text)
print(Terms)
```

Output:

```
['In', '2025', 'Dr', 'A.I.', 'Research', 'Lab', 'released', 'GPT', '5.0', 'with',
'1.75B', 'parameters', 'costing', '120.50M', 'It', 'outperforms', 'BERT',
'RoBERTa', 'etc', '...', 'and', 'supports', '100', 'languages']
```

3. Stopwords Removal

We remove common stop words using NLTK's corpus. Download nltk stopword list as follows:

```
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
StopWords = stopwords.words('english')
ExpReg = RegexpTokenizer(r'(?:[A-Z]\.)+|\d+(?:\.\d+)?[A-Za-z]*|\w+|\.{3}')

Terms = ExpReg.tokenize(Text)
TermsNoStop = [t for t in Terms if t.lower() not in StopWords]
print(TermsNoStop)
```

Output :

```
['2025', 'Dr', 'A.I.', 'Research', 'Lab', 'released', 'GPT', '5.0', '1.75B',
'parameters', 'costing', '120.50M', 'outperforms', 'BERT', 'RoBERTa', 'etc', '...',
'supports', '100', 'languages']
```

4. Terms Normalization (Stemming)

We normalize terms to reduce inflected forms to their root words using Porter Stemmer and Lancaster Stemmer

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
```

4.1. Porter Stemmer

```
Porter = PorterStemmer()
TermsPorter = [Porter.stem(t) for t in TermsNoStop]
print(TermsPorter)
```

Output:

```
['2025', 'dr', 'a.i.', 'research', 'lab', 'releas', 'gpt', '5.0', '1.75b',
'paramet', 'cost', '120.50m', 'outperform', 'bert', 'roberta', 'etc', '...',
'support', '100', 'languag']
```

4.2. Lancaster Stemmer

```
Lancaster = LancasterStemmer()
TermsLancaster = [Lancaster.stem(t) for t in TermsNoStop]
print(TermsLancaster)
```

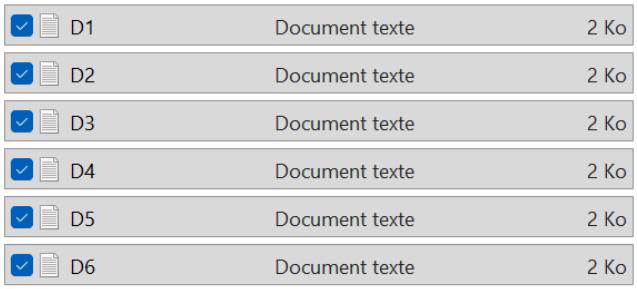
Output:

```
['2025', 'dr', 'a.i.', 'research', 'lab', 'releas', 'gpt', '5.0', '1.75b',
'paramet', 'cost', '120.50m', 'outperform', 'bert', 'robert', 'etc', '...',
'support', '100', 'langu']
```

EXERCISE 1

I. Read a Collection

Read the given collection containing several text files (D1.txt, D2.txt, D3.txt..D6) each with a different a text snippet.




```
# Read all documents from collection
documents = {}
for filename in os.listdir(collection_path):
    if filename.endswith(".txt"):
        doc_id = filename.split(".")[0]  # e.g., D1
        with open(os.path.join(collection_path, filename), "r", encoding="utf-8") as f:
            documents[doc_id] = f.read()
```

II. Build the Index

For each document:

1) Extract terms using both methods (split() and RegexpTokenizer).
2) Remove stop words.
3) Apply normalization (Porter and Lancaster).
4) Create the document.Txt term files, defined as follows:

   <Document number> <Term>

5) Create the inverted index files, defined as follows:

   <Term> <Document number>

DocTerm_Lancaster.txt

```
D1 10
D1 18
D1 24
D1 5
D1 9
D1 abl
D1 align
D1 approach
D1 art
D1 bas
D1 benchmark
D1 benefit
D1 bet
D1 context
D1 docu
D1 due
D1 ensembl
D1 evalu
D1 expery
D1 exploit
D1 feedback
D1 find
D1 four
D1 gain
D1 gen
D1 genqrensembl
D1 genqrensemblerf
D1 help
D1 improv
D1 incorp
D1 inh
D1 inspir
D1 instruct
D1 int
```

InvertedIndex_Porter.txt

```
10 D1
10 D4
10 D2
12 D4
175b D2
18 D1
1m D6
2019 D2
2020 D2
20b D2
24 D1
3.5 D6
4 D2
4.2 D2
5 D1
50x D2
9 D1
abil D1
abil D5
achiev D4
achiev D2
achiev D5
action D5
adapt D4
address D3
advanc D4
advantag D5
aggreg D4
aim D5
aim D4
align D1
allow D5
allow D3
alpaca D6
```

5. Create frequency dictionary

```python
TermFrequencies = {}
for term in TermsPorter:
    if term in TermFrequencies:
        TermFrequencies[term] += 1
    else:
        TermFrequencies[term] = 1

print(TermFrequencies)
```

Output:

```
{'2025': 1, 'dr': 1, 'a.i.': 1, 'research': 1, 'lab': 1, 'releas': 1,
'gpt': 1, '5.0': 1, '1.75b': 1, 'paramet': 1, 'cost': 1, '120.50m': 1,
'outperform': 1, 'bert': 1, 'roberta': 1, 'etc': 1, '...': 1,
'support': 1, '100': 1, 'languag': 1}
```

We can also use NLTK's FreqDist :

```
import nltk
TermFrequencies = nltk.FreqDist(TermsPorter)
print(TermFrequencies)
```

Output:

```
{'2025': 1, 'dr': 1, 'a.i.': 1, 'research': 1, 'lab': 1, 'releas': 1,
'gpt': 1, '5.0': 1, '1.75b': 1, 'paramet': 1, 'cost': 1, '120.50m': 1,
'outperform': 1, 'bert': 1, 'roberta': 1, 'etc': 1, '...': 1,
'support': 1, '100': 1, 'languag': 1}
<FreqDist with 20 samples and 20 outcomes>
```

Sorting the dictionary alphabetically:

```
import collections
TermFrequenciesSorted = collections.OrderedDict(sorted(TermFrequencies.items()))
print(TermFrequenciesSorted)
```

Output:

```
OrderedDict({'...': 1, '1.75b': 1, '100': 1, '120.50m': 1, '2025': 1,
'5.0': 1, 'a.i.': 1, 'bert': 1, 'cost': 1, 'dr': 1, 'etc': 1, 'gpt': 1,
'lab': 1, 'languag': 1, 'outperform': 1, 'paramet': 1, 'releas': 1,
'research': 1, 'roberta': 1, 'support': 1})
```

6. Weighting the Normalized Terms

Compute the weight for each term in the list. We use TF-IDF given as:

$$\text{weight}(t,d) = \frac{\text{freq}(t,d)}{\text{max freq in } d} \times \log_{10} \frac{N}{n_t + 1}$$

Where:

- $\text{freq}(t,d)$ = frequency of term t in document d
- $\text{max freq in } d$ = maximum frequency of any term in the document
- N = total number of documents in the collection
- n_t = number of documents containing term t

EXERCISE 2

I. Creating the indexes:

- Update the descriptor files as follows:

  <Document number> <Term> <Frequency> <Weight>

- Update the inverted index files as follows:

  <Term> <Document number> <Frequency> <Weight>

BASIC REGULAR EXPRESSION SYNTAX

Pattern	Meaning	Example Match
\w	Any alphanumeric character (A–Z, a–z, 0–9, and underscore)	A, 7, _
\d	Any digit	3, 45
\s	Any whitespace (space, tab, newline)	" " or \n
.	Any character except newline	a, %, Z
+	One or more repetitions	\d+ → 123, 42
*****	Zero or more repetitions	\w* → Hello, `` (empty)
?	Zero or one occurrence	colou?r → color, colour
[]	Character class (set of allowed characters)	[A-Z] → any capital letter
[^]	Negation (not these characters)	[^0-9] → anything except digits
()	Grouping or subexpression	(ab)+ → ab, abab
(?:)	Non-capturing group	Used for grouping without saving match
^	Start of string	^The → matches “The” at the beginning
$	End of string	end$ → matches “end” at the end
\.	Literal dot (.)	\.{3} → matches ...
\b	Word boundary	\bcat\b → matches “cat” but not “concatenate”