# TDT4117 Information Retrieval - Assignment 4

Andreas Bentzen Winje

Hermann Andreas Kran

# Contents

## List of Figures

## List of Listings

# Introduction

All notebooks have PDF versions for easy reading. See them for raw results. Else, all relevant results are shown in this report. All the files (images, listings, etc.) used in this report can be found in the submission zip.

Also, please try out our indexer given in the submission in `indexer.py` (just run the file with python, see section 2.2).

# 1 Task 1

All the code for this task can be found in `task1.ipynb`.

## 1.1 1a) Inverted index

Listing 1 shows an inverted index list for the given document.

```
"intelligent behavior people product mind mind itself more human brain does"

{
 'itself': {0: 1},
 'brain': {0: 1},
 'human': {0: 1},
 'intelligent': {0: 1},
 'people': {0: 1},
 'mind': {0: 2},
 'product': {0: 1},
 'more': {0: 1},
 'behavior': {0: 1},
 'does': {0: 1}
 }
```

**Listing 1:** The document inverted index.

## 1.2 1b) Block addressing

Listing 2 shows the output from creating an inverted index list with block addressing with `block-size=3`. Here we can also see the generated blocks and their indices.

```
"intelligent behavior people product mind mind itself more human brain does"

Blocks:
0 ['intelligent', 'behavior', 'people']
1 ['product', 'mind', 'mind']
2 ['itself', 'more', 'human']
3 ['brain', 'does']

Index list:
{
'itself': {0: [2]},
 'brain': {0: [3]},
 'human': {0: [2]},
 'intelligent': {0: [0]},
 'people': {0: [0]},
 'mind': {0: [1]},
 'product': {0: [1]},
 'more': {0: [2]},
 'behavior': {0: [0]},
 'does': {0: [3]}
 }
```

**Listing 2:** Indexing using block addressing.

## 1.3 1c) Suffix tree

In listing 3, a partial vocabulary suffix tree can be seen for the given text. Here we assume "vocabulary" means word-level instead of character-level suffixes, and "partial" means without stopwords. "$" marks end condition. Here, unary paths ending in a leaf node are removed to decrease the amount of nodes, as suggested in the lecture.

```
     1           2        3        4      5    6     7      8     9     10    11  12
intelligent behavior people product mind mind itself more human brain does $

(root)
    +-$-(12)
    +-does-(11)
    +-brain-(10)
    +-human-(9)
    +-more-(8)
    +-itself-(7)
    +-mind
    |   +-itself-(6)
    |   +-mind-(5)
    +-product-(4)
    +-people-(3)
    +-behavior-(2)
    +-intelligent-(1)
```

**Listing 3:** The suffix tree.

## 1.4 1d) Indexing small corpus

The inverted index list for the corpus in task 1d can be seen in listing 4.

```
{
 'total': {1: 1},
 'big': {2: 1},
 'pretty': {1: 1},
 'pieces': {3: 1},
 'understand': {4: 1},
 'mystery': {2: 1},
 'even': {0: 1},
 'brain': {0: 1},
 'ten': {1: 1},
 'put': {3: 1},
 'puzzle': {2: 1},
 'human': {0: 1},
 'much': {0: 1, 1: 1, 3: 1},
 'ago': {1: 1},
 'more': {0: 1},
 'all': {4: 1},
 'although': {0: 1},
 'jigsaw': {2: 1},
 'remains': {1: 1},
 'years': {1: 1},
 'thinking': {1: 1},
 'see': {2: 1},
 'know': {0: 1},
 'many': {2: 1},
 'engages': {1: 1},
 'together': {3: 1}
 }
```

**Listing 4:** The corpus inverted index list.

# 2 Task 2

## 2.1 2a) The ELK-stack

Elastic Stack (ELK Stack) is a search platform that is compromised of Elasticsearch, Kibana, Beats, and Logstash and more. The amount of features the Stack is enormous, but in the broad strokes ELK Stack assists with ingesting, analyzing, searching, and visualize all types of data at scale.

Lucene is an Apache framework that provides an open search software. The software is often released as a core search library named Lucene core, as well as PyLucene for Python. Some powerful utilities of Lucene core are indexing, search features, spellchecking and hit highlighting.

As for how ELK Stack utilizes Lucene it is easier to illustrate that with the application that compromises ELK Stack than the stack. Elasticsearch utilizes Lucene's search library for full text capabilities. Logstash utilizes Lucene to perform text analysis and indexing of log data. Kibana has the Lucene query syntax as an alternative to the Kibana Query Language. The reasoning for opting to choose Lucene query syntax it that it allows for more advance Lucene features such as regular expressions and fuzzy term matching. The downside is that unlike the Kibana Query Language it can't search nested objects or scripted fields. The main purpose of both query forms is to filter data.

## 2.2 2b) Custom indexer

In fig. 1, one can see a diagram of how our indexer was imagined to work.

The final implementation features indexing (obviously), nested boolean parsing (parsing of boolean expression WITH parentheses) and ranking with TF-IDF (just comparing raw TF-IDF score). The indexer supports boolean operators: AND, OR, and NOT (also written as '-'). You can test our indexer by simply running ´python3 indexer.py´.

You can for example try to search for "(fox AND brown AND lazy) OR (claim -morning)" and see that you get the correct result. Still, it is not very well tested, so one might find some errors in the boolean parsing, but it works for very basic, general tasks.

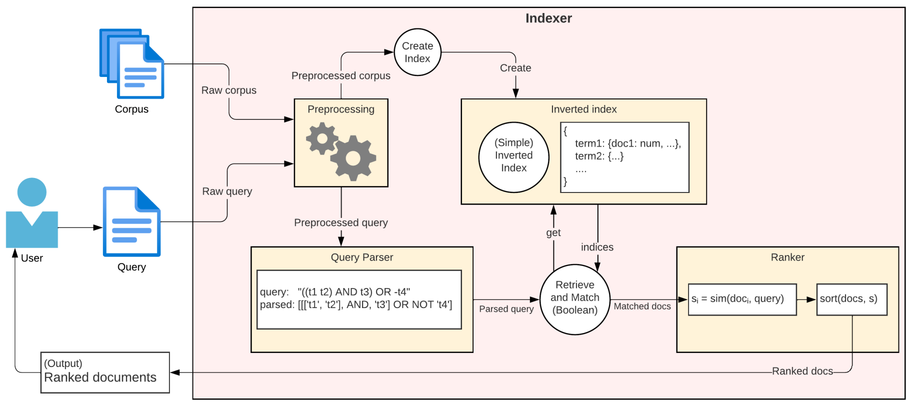


**Figure 1:** Diagram for custom indexer.

### 2.3 2c) Indexing

See notebooks or notebook PDFs.

### 2.4 2d) Some searching

Results for the queries 'claim', 'claim*' and 'claims of duty' can be seen in listing 5, listing 6, and listing 7, respectively.

```
query: claim

ElasticSearch:
Results (1):
ID: 1, Score: 1.5535183, Content: One morning, when Gregor Samsa woke from troubled
dreams, he found himself transformed in his bed in

Custom (ours):
Results (2):
ID: 5, Score: 4.7549, Content: But I must explain to you how all this mistaken idea
of denouncing pleasure and praising pain was bo
ID: 1, Score: 1.5850, Content: One morning, when Gregor Samsa woke from troubled
dreams, he found himself transformed in his bed in
```

**Listing 5:** Results for 'claim'

```
query: claim*

ElasticSearch:
Results (2):
ID: 1, Score: 1.0, Content: One morning, when Gregor Samsa woke from troubled
dreams, he found himself transformed in his bed in
ID: 5, Score: 1.0, Content: But I must explain to you how all this mistaken idea of
denouncing pleasure and praising pain was bo

Custom (ours):
Results (2):
ID: 5, Score: 4.7549, Content: But I must explain to you how all this mistaken idea
of denouncing pleasure and praising pain was bo
ID: 1, Score: 1.5850, Content: One morning, when Gregor Samsa woke from troubled
dreams, he found himself transformed in his bed in
```

**Listing 6:** Results for 'claim*'

```
query: claims of duty

ElasticSearch:
Results (6):
ID: 5, Score: 5.4214234, Content: But I must explain to you how all this mistaken
idea of denouncing pleasure and praising pain was bo
ID: 0, Score: 0.15978271, Content: A wonderful serenity has taken possession of my
entire soul, like these sweet mornings of spring whi
ID: 3, Score: 0.15705192, Content: Far far away, behind the word mountains, far
from the countries Vokalia and Consonantia, there live
ID: 4, Score: 0.15643656, Content: The European languages are members of the same
family. Their separate existence is a myth. For scien
ID: 1, Score: 0.15394345, Content: One morning, when Gregor Samsa woke from
troubled dreams, he found himself transformed in his bed in
ID: 2, Score: 0.14411825, Content: The quick, brown fox jumps over a lazy dog. DJs
flock by when MTV ax quiz prog. Junk MTV quiz graced

Custom (ours):
Results (2):
ID: 5, Score: 20.2647, Content: But I must explain to you how all this mistaken
idea of denouncing pleasure and praising pain was bo
ID: 1, Score: 1.5850, Content: One morning, when Gregor Samsa woke from troubled
dreams, he found himself transformed in his bed in
```

**Listing 7:** Results for 'claims of duty'

## 2.5 2e) Dive into a search query

**How is "claims of duty" handled by the two indexers (ours and ELK)?**

Our implementation would simply take each term, preprocess them, find all the documents that correspond to each term, and then add all of the documents together. Then the documents are ranked by TF-IDF score. Since there are no operators in the queries, we also don't have any "intersectioning" of documents.

ELK uses a similar process, where it tokenizes and preprocesses the terms, then finds all the documents for each term and accumulates them. ELK also provides document ranking using the BM25 model.

**Is there a similar query that provides the same result in your implementation and in the ELK stack?**

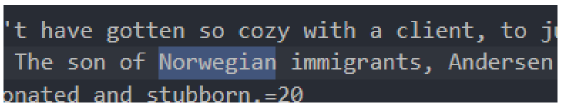
The notebook crashed before this was tested, and it takes about 40-ish minutes to load and index everything (assuming it doesn't crash again), but it seems that the two indexers generally get similar results.

**Do we get the expected results?**

With our indexer, we get only the two documents containing (the stem of) either "claims" or "duty", which are the two interesting terms in this query (because we remove stopwords), while the ELK stack retrieved all documents, probably because of the term "of". In this case, our indexer technically has better precision. This is probably because it is so few documents, and for larger datasets ELK would presumably have a better precision and performance than ours. The results we got are expected, anyway.

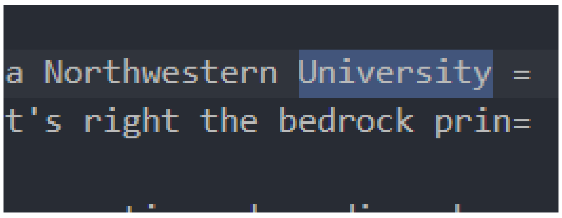
## 2.6 2f) Indexing 200'000 documents

Results from the boolean NTNU query can be seen in listing 8. Here we see that Elasticsearch got none results, while we got one result with our own indexer. After manual analysis, it was found that this one email had nothing to do with NTNU, see images in fig. 2. The results from the non-boolean query can be seen in listing 9, where we can see that Elasticsearch had 10 results, while we had 8641. This difference could be due to Elasticsearch having better filtering of irrelevant documents (as we have none other than boolean and term matching), but it is not sure if this impacts the recall. Please see the notebook PDFs for full output.



```
't have gotten so cozy with a client, to j
 The son of Norwegian immigrants, Andersen
onated and stubborn.=20
```

**(a)** The one hit for "Norwegian".



```
a Northwestern University =
t's right the bedrock prin=
```

**(b)** The one hit for "University"

**Figure 2:** Images from the email found in task 2f. We see that none of the terms are related to NTNU.

```
query = Norwegian AND University AND Science AND Technology

ElasticSearch:
Results (0):
Elapsed 0.00699925422668457s

Custom (ours):
Results (1):
ID: 122725, Score: 96.2405, Content:
Message-ID:<15111552.1075841109495.JavaMail.evans@thyme>Date: Wed, 30 Jan 2002
07:22:55 -0800 (PST
Elapsed 0.002999544143676758s
```

**Listing 8:** Results for NTNU boolean query

```
query = Norwegian University Science Technology

ElasticSearch:
Results (10):
ID: 83815, Score: 20.404312, Content: ...
ID: 89895, Score: 20.404312, Content: ...
ID: 83809, Score: 19.514158, Content: ...
ID: 89900, Score: 19.514158, Content: ...
ID: 12286, Score: 18.755344, Content: ...
ID: 55651, Score: 18.755344, Content: ...
ID: 64118, Score: 18.755344, Content: ...
ID: 83644, Score: 18.454838, Content: ...
ID: 97006, Score: 18.111553, Content: ...
ID: 77685, Score: 16.287563, Content: ...
Elapsed 0.017000198364257812s

Custom (ours):
Results (8641):
ID: 96374, Score: 380.3556, Content: ...
ID: 96380, Score: 276.7993, Content: ...
ID: 96375, Score: 276.7993, Content: ...
ID: 123200, Score: 172.6853, Content: ...
ID: 123178, Score: 167.9916, Content: ...
ID: 97004, Score: 129.8882, Content: ...
ID: 112562, Score: 129.6079, Content: ...
ID: 13984, Score: 129.6079, Content: ...
ID: 112595, Score: 120.0146, Content: ...
...
ID: 163848, Score: 5.2880, Content: ...
ID: 0, Score: 5.2880, Content: ...
Elapsed 1.0641894340515137s
```

**Listing 9:** Results for NTNU normal query. Content is not shown here.