

The Tenth International Conference on Advances in Databases, Knowledge, and Data Applications

Mai 20 - 24, 2018 - Nice/France

How to build a Search-Engine with Common Unix-Tools

Andreas Schmidt

(1)

**Department of Informatics and
Business Information Systems
University of Applied Sciences Karlsruhe
Germany**

(2)

**Institute for Automation and Applied Informatics
Karlsruhe Institute of Technologie
Germany**

Resources available

<http://www.smiffy.de/dbkda-2018/>¹

- Slideset
- Exercises
- Command refcard

1. all materials copyright, 2018 by andreas schmidt

Outlook

- General Architecture of an IR-System
 - Naive Search
 - Boolean Search
 - Vector Space Model
 - Inverted Index
 - Query Processing
- Overview of useful Unix Tools
- Implementation Aspects
- Summary

+ 2 hands on exercises

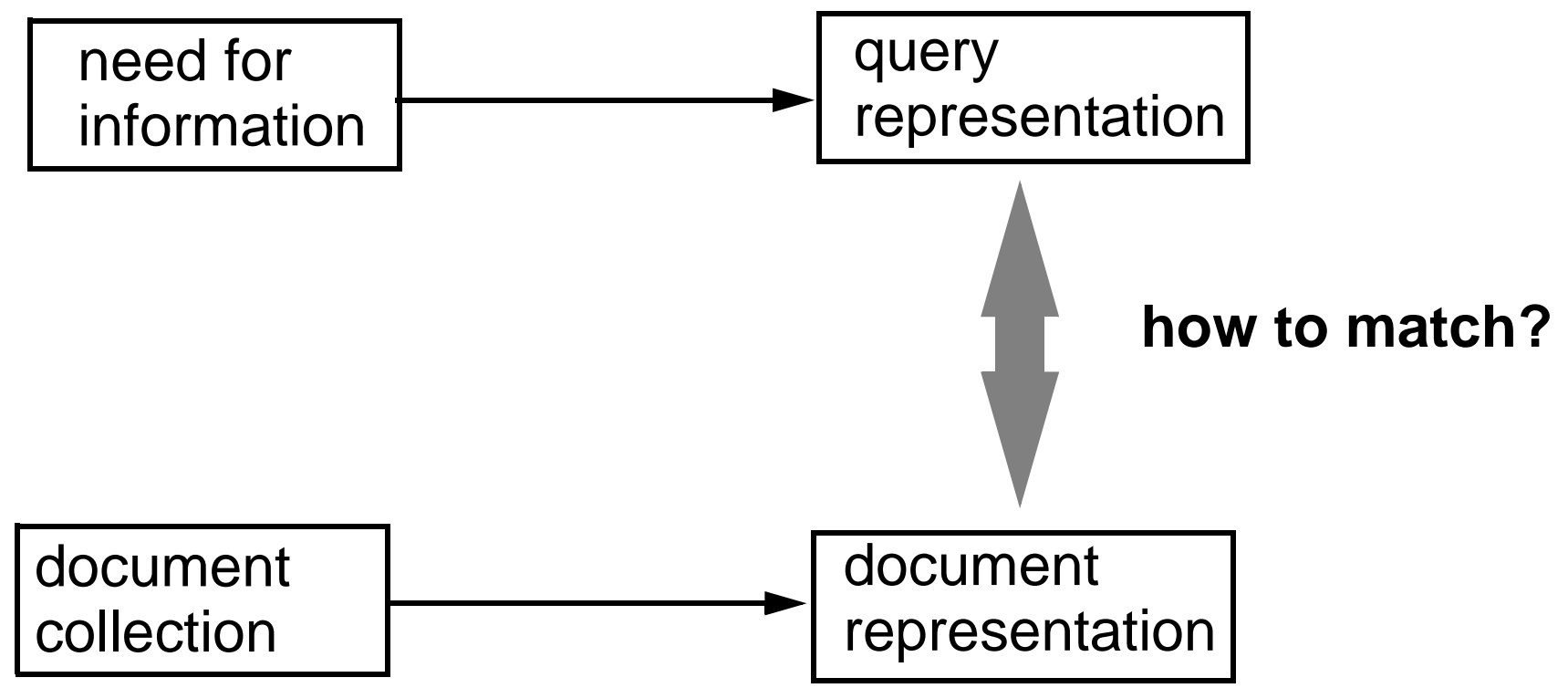
- Text analytics
- Building an Inverted Index & Query processing

What is Information Retrieval ?

Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need (usually a query) from within large collections (usually stored on computers).

[Manning et al., 2008]

What is Information Retrieval ?



Keyword Search

- Given:
 - Number of Keywords
 - Document collection
- Result:
 - All documents in the collection, containing the keywords
 - (ranked by relevance)

Naive Approach

- Iterate over all documents d in document collection
 - For each document d , iterate all words w and check, if all the given keywords appear in this document
 - if yes, add document to result set
- Output result set

- Extensions/Variants
 - Ranking
 - multiword terms (***New York***)
 - 'near' semantic (i.e. ***trump*** near ***russia***)

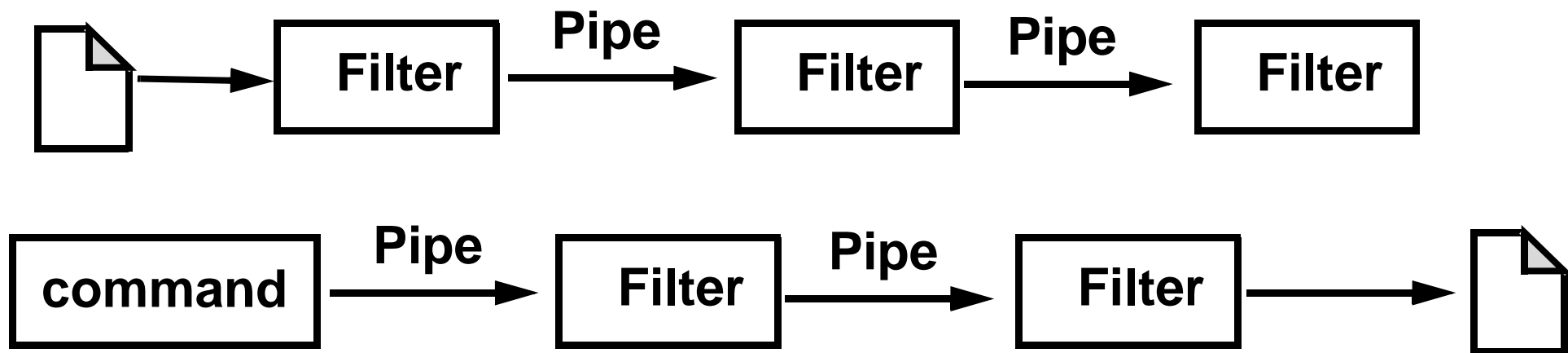
see examples later

Naive Approach

- Advantages:
 - Good for small and medium size datasets
 - No index need to be build before
 - Speed: up to 2 GB per second
- Disadvantages:
 - Not feasible for larger datasets (100 GB: 1 min. per search request)
 - Potentiell additional preprocessing for each query needed
- Implementation with the shell using the commands
 - grep (main work)
 - tr, sort, sed, uniq, sort, cut, join (pre-/postprocessing)

Data Processing with the Shell

- Architectural Pattern: Pipes and Filters (Douglas McIlroy, 1973)
- Data exchange between processes
- Loose coupling
- POSIX Standard
- Filter represent data-sources and data-sinks



Shell commandos in the Linux/Unix/Cygwin Environment

- Input-/Output channels
 - Standardinput (STDIN)
 - Standardoutput (STDOUT)
 - Standarderror (STDERR)
- In-/Output Redirection
 - > : Redirect Standardoutput (into file)
 - < : Redirect Standardinput (from file)
 - 2> : Redirect Standarderror (into file)
 - >> : Redirect Standardoutput (append into file)
 - | : Pipe operator: Connect Standardoutput of a command with Standardinput of the next command

- Example:

```
cut -d, -f1 city.csv | sort | uniq -c | sort -nr | \
awk '$1>1 {print $2}' > result.txt
```

Overview of Commands used in this Tutorial

- **grep**: print lines matching a pattern
- **tr**: translate or delete characters
- **comm**: compare two sorted files line by line
- **uniq**: report or omit repeated lines
- **join**: join lines of two files on a common field
- **cat**: concatenate files and print on the standard output
- **sort**: sort lines of text files
- **sed**: stream editor for filtering and transforming text
- **awk**: pattern scanning and processing language
- **wc**: Counts words, bytes, lines
- **cut**: Extracts columns from a file
- ...

General comment

- Most of the commands accept the input from file or from STDIN. If no (or not enough) input files are given, it is expected that the input comes from STDIN

```
head -n4 my-file.txt  
cat -n my-file.txt | head -n4
```

- Most of the commands have a lot of options which couldn't be explained in detail. To get an overview of the possibilities of a command, simply type

```
man command
```

- Example:

```
man head
```

```
 /cygdrive/c/Users/scan0004/Dropbox/dbkda-2017/tutorial
HEAD<1>                                User Commands                                HEAD<1>
NAME
  head - output the first part of files
SYNOPSIS
  head [OPTION]... [FILE]...
DESCRIPTION
  Print the first 10 lines of each FILE to standard output.  With more
  than one FILE, precede each with a header giving the file name.

  With no FILE, or when FILE is -, read standard input.

  Mandatory arguments to long options are mandatory for short options
  too.

  -c, --bytes=[-]NUM
        print the first NUM bytes of each file; with the leading '-',
        print all but the last NUM bytes of each file
  -n, --lines=[-]NUM
        print the first NUM lines instead of the first 10; with the
        leading '-', print all but the last NUM lines of each file
  -q, --quiet, --silent
        never print headers giving file names
  -v, --verbose
        always print headers giving file names
  -z, --zero-terminated
        line delimiter is NUL, not newline
  --help display this help and exit
  --version
        output version information and exit
Manual page head(1) line 1 (press h for help or q to quit)
```

Online Search using grep

- Multi line phrase match:

```
$ cat multiline-match.txt
```

```
This is an example of a multi line  
match. In this case the phrase 'multi line match'  
should be found, even if the words appear in  
separate lines.
```

```
$ cat multiline-match.txt | tr '\n' ' ' | grep -o 'multi line  
match'
```

```
multi line match
```


```
multi line match
```

Search for 'teaching' near 'students'

```
$ less papers/1273.txt  
The teaching is created with  
students despite the number  
which is about one hundred and  
fifty. The lecturer asks ques-  
tions related to the study ...
```

```
$ tr -cs < papers/1273.txt \  
    'A-Za-z' '\n' | less  
The  
teaching  
is  
created  
with  
students  
despite  
the  
number  
which  
is  
about  
one  
hundred
```

n-words distance



Search for 'teaching' near 'students'

```
$ tr < ../proceedings/papers/1273.txt -cs 'A-Za-z0-9' '\n' | \  
  grep -5 teaching|less
```

```
--
```

```
ask
```

```
questions
```

```
fig
```

```
1
```

```
The
```

```
teaching
```

```
is
```

```
created
```

```
with
```

```
students
```

```
despite
```

```
--
```

```
the
```

grep with additional context
(n-lines before/after the match)

Search for 'teaching' near 'students'

```
export MAX_DIST=5
export TXT_DOCS=papers/*.txt
rm -f result.txt
for f in $TXT_DOCS; do
    tr -sc '[A-Za-z]+' '\n' < $f | tr 'A-Z' 'a-z' > $f.2;
    grep -H -MAX_DIST -i teaching $f.2 | grep -i students | \
        sed -r 's#\.\.2[-:][a-z]+###' >> result.txt
done
rm -f $TXT_DOCS.2
uniq -c result.txt | sort -nr|less
```

- Output (sorted by decreasing relevance)

```
11 ../proceedings/papers/1273.txt
 8 ../proceedings/papers/1442.txt
 3 ../proceedings/papers/1351.txt
 3 ../proceedings/papers/1250.txt
 3 ../proceedings/papers/1210.txt
 3 ../proceedings/papers/1140.txt
 3 ../proceedings/papers/1121.txt
 3 ../proceedings/papers/1114.txt
 2 ../proceedings/papers/1504.txt
 2 ../proceedings/papers/1464.txt
 2 ../proceedings/papers/1303.txt
 2 ../proceedings/papers/1298.txt
...
```

Exercise I

Download Exercise 1 from

<http://www.smiffy.de/dbkda-2018/IR-exercise-1.pdf>

with solutions:

<http://www.smiffy.de/dbkda-2018/IR-exercise-1-solution.pdf>

Online Search - Discussion

- Good for small and medium size datasets
- No additional datastructure/tools/systems needed
- Disadvantage: Speed to perform online search, if document base is big
- Alternate Approach:
 - Build an appropriate datastructure for fast retrieval (offline)
 - Query the datastructure

Inverted Index

- Document-set datastructure:

...

`documentn -> (sorted list of words in documentn)`

`documentn+1 -> (sorted list of words in documentn+1)`

...

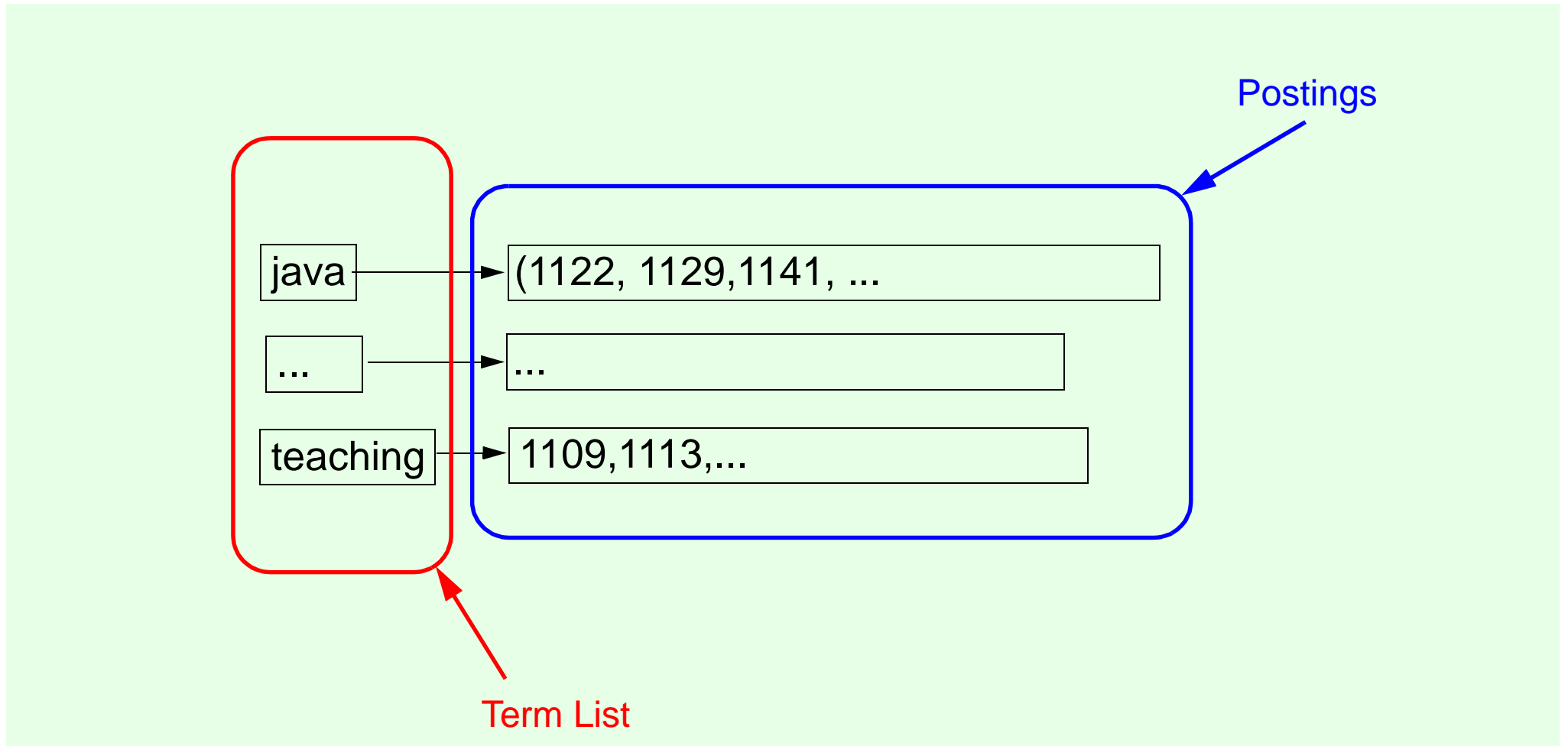
- Needed: Alternate datastructure, to store information, in which documents a word appears
- General structure (words sorted alphabetically):

...

`wordn -> (sorted list of documents containing wordn)`

`wordn+1 -> (sorted list of documents containing wordn+1)`

...



Principles

- Each document is considered as a set of words
- Typically, the term list is stored in memory with a link to the document list (postings) on disk
- term list and posting list are stored sorted
- Disadvantage: No ranking possible

some statistics ...

- Conference Proceedings (IEEE EDUCON 2018)
- 300 papers
- 1473809 words altogether
- 47288 different words (without stemming)
- 38033 different words (with stemming)
- avg: ~ 4913 words/paper
- avg: ~ 1139 different words/paper (without stemming)
- avg: ~7.5 documents per word

- Compare the amount of memory to be read using grep and using an inverted index
 - grep: 1,920,421 bytes
 - Inverted Index: Access word list + 7.5 document identifiers (on average)

General Indexer Principle (Blocked, Sort-Based Indexing)

- For performance reason, each document-path is assigned a numeric ID, which is then used during the previously described process
- Collect all document-term pairs
- Sort pairs by their term (first criteria) and document identifier (second criteria)
- Collect all pairs (term, document) with the same term into a single entry of the inverted list.

Modeling an Inverted Index using Files

- Entries (words) represent files
- Postings (document IDs) are represented by the content (one document-ID per line)
- Postings are sorted by ID
- Example:

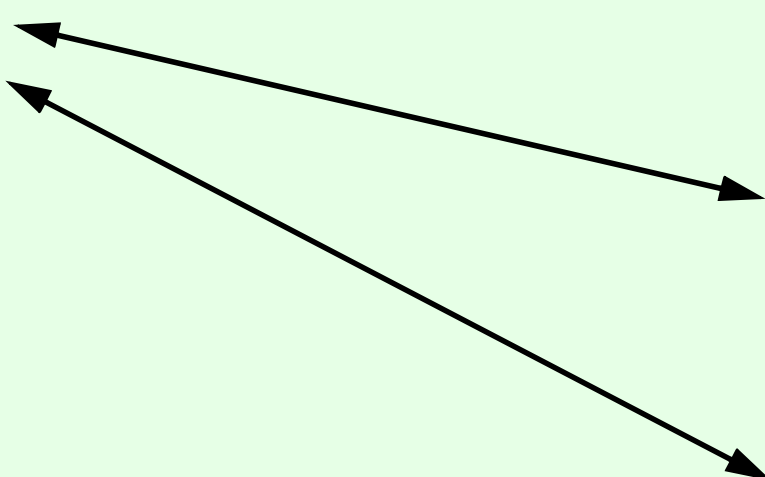
```
$ cat invIndex/drivers.txt  
01207  
01226  
01234  
01242  
01279  
01363  
...
```

- Eventually using subfolders to keep number of files per directory small

How to Process a Query

- Example: Search for documents containing the word **didactic** and **student**

```
$ cat invIndex/didactic.idx      $ cat invIndex/student.idx
01121                            01109
01136                            01114
01157                            01118
01162                            01121
01176                            01124
01183                            01127
01197                            01129
01203                            01131
01214                            01136
...                               ...
```



- Result:
Documents 01121, 01136, ...

- Example: Search for documents containing the word **student**, but not **teaching**

\$ invIndex/student.idx		\$ cat invIndex/teaching.idx
01109	←-----→	01109
01114	←-----→	01113
01118	←-----→	01114
01121	←-----→	01118
01124	←-----→	01120
01127	←-----→	01122
01129	←-----→	01124
01131	←-----→	01127
01136	←-----→	01130
...		...

- Result:

Documents **01121**, **01129**, ...

Step 1: Transform all PDF-Documents to Text Format

```
# We assume, that we have a number of pdf documents to index.  
# In this case, we use xpdf [1]
```

```
export XPDF=c:/Programme/xpdf-3.0/bin64/pdftotext.exe  
export PDF_DOCS=papers/*.pdf
```

```
#!/usr/bin/bash  
for f in $PDF_DOCS; do  
    $XPDF $f;  
done
```

```
# Result: For every pdf document, a txt-document with the same  
# name is generated in the same directory
```

[1] <https://www.xpdfreader.com/>

Step 2: Assign each File a Unique Document ID

- List all documents and add a line number

```
ls papers/*.txt | cat -n
  1  papers/1109.txt
  2  papers/1112.txt
  3  papers/1113.txt
  4  papers/1114.txt
  ...
```

- Format the output

```
ls papers/*.txt | cat -n | awk '{printf "%04d:%s\n", $1, $2}' \
> file-id.map
0001  papers/1109.txt
0002  papers/1112.txt
0003  papers/1113.txt
0004  papers/1114.txt
  ...
```

Step 3: Extract different words from a Document

```
# Tokenize the text document.  
# We look for the regex-pattern [A-Za-z]+ (1-n characters) and  
# print them to STDOUT. With the -o option, we only print the  
# matching part. Because multiple files are given, grep adds the  
# filename in front of each word.  
#  
$ grep -o -E '[A-Za-z]+' papers/*.txt  
papers/1109.txt:The  
papers/1109.txt:influence  
papers/1109.txt:of  
papers/1109.txt:class  
papers/1109.txt:attendance  
papers/1109.txt:on  
papers/1109.txt:the  
papers/1109.txt:throughput  
...
```

Step 4: Lowercase all, sort by term

- Lowercase everything

```
grep -o -E '[A-Za-z]+' papers/*.txt | tr 'A-Z' 'a-z' > doc-term.txt  
papers/1109.txt:the  
papers/1109.txt:influence  
papers/1109.txt:of  
papers/1109.txt:class  
papers/1109.txt:attendance  
...
```

- Sort by term (first criteria) and by filename (second criteria)

```
sort -t: -k2,2 -k1,1 doc-term.txt  
papers/1109.txt:a  
papers/1109.txt:a  
...  
papers/1112.txt:a  
papers/1112.txt:a  
...
```


Step 5: Remove Duplicates

- Remove Duplicates

```
sort -t: -k2,2 -k1,1 doc-term.txt |uniq > term-file.txt
```

```
papers/1109.txt:a
```

```
papers/1112.txt:a
```

```
papers/1113.txt:a
```

```
papers/1114.txt:a
```

```
...
```

```
papers/247.txt:a
```

```
papers/1236.txt:aaai
```

```
papers/1330.txt:aaai
```

```
...
```

Step 6: Replace document-name by ID

```
$ sort -t: term-file.txt | head -n5
papers/1109.txt:a
papers/1109.txt:about
papers/1109.txt:absenteeism
papers/1109.txt:abstract
papers/1109.txt:ac

$ head -n5 file-id.map
0001:papers/1109.txt
0002:papers/1112.txt
0003:papers/1113.txt
0004:papers/1114.txt
0005:papers/1118.txt
```

```
$ sort -t: term-file.txt | join -11 -22 -t: - file-id.map -o1.2,2.1
a:0001
about:0001
absenteeism:0001
abstract:0001
ac:0001
academia:0001
academic:0001
```

Diagram annotations:

- Red arrows point from the `-11` and `-22` options to the `join` command, labeled **join-columns**.
- Red arrows point from the `-t:` option to the `join` command, labeled **column separator**.
- Red arrows point from the `-o1.2,2.1` option to the `join` command, labeled **output columns**.

- Sort by Term

```
$ sort -t: term-file.txt | join -11 -22 -t: - file-id.map \  
-o1.2,2.1 | sort -t: -k1,2 | tee invIndex.idx
```

a:0001

a:0002

...

a:0299

a:0300

aaai:0086

aaai:0140

aaai:0143

...

Next step:

- Write all entries with same term in a single file

awk-Intro needed

Generate file-based entry

- File: write-inverted-index-to-file.awk

```
{  
  if ($1 != last) {  
    print $2 > DIR"/"$1".txt"  
    last = $1  
  } else {  
    print $2 >> DIR"/"$1".txt"  
  }  
}
```

create a new file

append to file

- Execution:

```
awk -F: -v DIR=d:/data/invIndex \  
-f write-inverted-index-to-file.awk invIndex.idx
```

Generated Inverted Index:

```
$ invIndex/teaching.txt
```

```
0001
```

```
0003
```

```
0004
```

```
0005
```

```
0006
```

```
0007
```

```
...
```

```
$ ls invIndex/tea*
```

```
d:/data/invIndex/tea.txt
```

```
d:/data/invIndex/teach.txt
```

```
d:/data/invIndex/teachable.txt
```

```
d:/data/invIndex/teacher.txt
```

```
d:/data/invIndex/teachercentered.txt
```

```
d:/data/invIndex/teacherfocused.txt
```

```
d:/data/invIndex/teacherled.txt
```

```
...
```

Example Query:

- Search for documents, containing the keywords, **teaching** and **students**

```
$ comm -1 -2 ./teaching.txt ./students.txt | \  
  join -11 -21 - file-id.map -t: -o2.2
```

```
papers/1109.txt
```

```
papers/1113.txt
```

```
papers/1114.txt
```

```
papers/1118.txt
```

```
papers/1121.txt
```

```
...
```

Boolean Search - Discussion

- Advantages/Disadvantages
 - + Easy to implement
 - - Small index (fast)
 - - No ranking possible

Ranking of Results

- Possible criteria:
 - The number of times, a word appears in a text
 - The relevance of the word ('whatever' vs. 'cambridge')
- Approach:
 - Store for each word, the number of times it appears in a document (Term Frequency - $tf_{\text{term}, \text{doc}}$)
 - Store for each word, in how many document it appears (Document Frequency - df_t)
- Ranking:
 - If a term appears more often in document A than in document B, document A is considered more relevant for the query
 - Terms which appears in a smaller number of documents have a higher weight

tf*idf (TFIDF)

- term frequency–inverse document frequency
- Measure how important a word for a document is
- $tf_{t,d}$: Measure, how many times a term t appears inside a document d (typically normalized)
- idf_t : Measure to distinguish important from unimportant terms
- Definition:
 - df_t : In how many documents of the collection D does term t appear
 - $idf_t = \log(N/df_t)$ # high for rare terms, low for frequent terms
 - N : Number of documents in collection D
- Composite weight for each term in each document:
 - $tfidf_{t,D} = tf_{t,d} * idf_t$

tf*idf (TFIDF)

- Score of a document with respect to a query with terms q (q_1, \dots, q_n).

$$\text{score}(q, d) = \sum_{t \in q} \text{tfidf}_{t, doc}$$

- Vector Space model
- Idea: Represent each document as a vector V in a n -dimensional vector space
- Dimensions are spanned by terms in the document collection
- Similarity of two documents is calculated by the angle between vector representation of each document

$$\text{sim}(d_1, d_2) = v(d_1) / |v(d_1)| * v(d_2) / |v(d_2)|$$

$$\text{Dot product: } v(d_1) * v(d_2) = \sum_{i=1..M} (d_{1i} * d_{2i})$$

$$\text{Euclidian length: } |v(d_1)| = \sqrt{\sum_{i=1..M} (v_i * v_i)}$$

- Query is also a vector ...

Index Structure

- **Resume:** We have to store additional information in our index

java → 33, [(1122,1), (1129,1), (1141,1), ...]

... → ...

teaching → 258, [(1109,2), (1113,2), ...]

$df_{teaching}$

$tf_{teaching,doc_1109}$

- tf-idfIndex/java.txt

```
# occurrence: 33 docs
papers/1122.txt:1
papers/1129.txt:1
papers/1141.txt:1
...
```

- ...

- tf-idfIndex/teaching.txt

```
# occurrence: 238 docs
papers/1109.txt:2
papers/1113.txt:2
papers/1114.txt:121
papers/1118.txt:27
...
```

- Sort by term (first criteria) and by filename (second criteria)

```
sort -t: -k2,2 -k1,1 doc-term.txt  
papers/1109.txt:a  
papers/1109.txt:a  
...  
papers/1112.txt:a  
papers/1112.txt:a
```

- Remove duplicates and count them

```
$ sort -t: -k2,2 -k1,1 doc-term.txt |uniq -c|head  
  69 papers/1109.txt:a  
 631 papers/1112.txt:a  
   42 papers/1113.txt:a  
  125 papers/1114.txt:a  
  119 papers/1118.txt:a  
   66 papers/1121.txt:a
```

↑
count

- Minor formatting issues ...

```
$ sort -t: -k2,2 -k1,1 doc-term.txt |uniq -c| \  
    awk -F' ' '{printf "%s:%d\n", $2, $1}'> invIndexTfIdf.idx
```

```
papers/1109.txt:a:69  
papers/1112.txt:a:631  
papers/1113.txt:a:42  
papers/1114.txt:a:125  
papers/1118.txt:a:119  
papers/1121.txt:a:66  
papers/1122.txt:a:279  
papers/1124.txt:a:138  
papers/1126.txt:a:177  
papers/1127.txt:a:100
```

```
# replace document name by ID is ommited here (see boolean search)
```

Distribute in Multiple Files (One File per Term)

```
$ awk -F: -v DIR=d:/data/tfIdfInvIndex \  
    -f write-inverted-tfidf-index-to-file.awk \  
    invIndexTfIdf.idx
```

- write-inverted-tfidf-index-to-file.awk

```
{  
    if ($1 != last) {  
        print $2":"$3 > DIR"/"$1".idx"  
        last = $1  
    } else {  
        print $2":"$3 >> DIR"/"$1".idx"  
    }  
}
```

- ... and what's about the Document Frequency (df_{term}) ?
- Improve write-inverted-tfidf-index-to-file.awk to also write out the Document Frequency df_{term}

Extended awk-script (document frequency)

```
{
  if ($1 != last) {
    if (last!="")
      print count > DIR"/"last".df"
    print $2":"$3 > DIR"/"$1".idx"
    last = $1
    count = 1
  } else {
    print $2":"$3 >> DIR"/"$1".idx"
    count++
  }
}
END {
  print $2":"$3 > DIR"/"last".df"
}
```

← for the last entry in file

Show Index:

```
$ head tfIdfInvIndex/java.idx
```

```
papers/1122.txt:1  
papers/1129.txt:1  
papers/1141.txt:1  
papers/1149.txt:4  
papers/1171.txt:6  
papers/1191.txt:4  
papers/1203.txt:1  
papers/1218.txt:8  
papers/1346.txt:1  
papers/1359.txt:1
```

```
$ head tfIdfInvIndex/java.df
```

```
33
```

```
$ head tfIdfInvIndex2/java.idx
```

```
0007:1  
0011:1  
0018:1  
0024:4  
0039:6  
0053:4  
0060:1  
0069:8  
0147:1  
0153:1
```

```
$ head tfIdfInvIndex2/java.df
```

```
33
```

Example Query:

```
export DF_JAVA=$(cat d:/data/tfIDFInvIndex/java.df)
export DF_PHP=$(cat d:/data/tfIDFInvIndex/php.df)
export N=$(ls papers/*.txt | wc -l)
join -11 -21 -t:      d:/data/tfIdfInvIndex/php.idx \
      -o1.1,1.2,2.2 d:/data/tfIdfInvIndex/java.idx > result.txt
export LC_ALL=C && awk -F: -v DF1=$DF_JAVA -v DF2=$DF_PHP -v N=$N \
      -f ranking.awk result.txt | \
      sort -t: -k2,2nr
papers/1587.txt:66.5875
papers/1441.txt:44.4532
papers/1218.txt:33.3553
papers/1466.txt:20.1116
papers/1149.txt:11.1595
papers/1517.txt:6.68337
papers/1457.txt:6.6526
...
```

Calculation of tf*idf values

- ranking.awk

```
{  
    tf1 = $2  
    tf2 = $3  
    idf1 = log(N/DF1)*tf1  
    idf2 = log(N/DF2)*tf2  
    print $1":idf1+idf2  
}
```

file identifier



Exercise II

Download Exercise 2 from

<http://www.smiffy.de/dbkda-2018/IR-exercise-2.pdf>

with solutions:

<http://www.smiffy.de/dbkda-2018/IR-exercise-2-solution.pdf>

Phrase Match - Positional Indexes

- For phrase queries such like „Dead men don't wear plaid“ we also need information about the position of a word in a file

perl → 3, [<1122, 4, (19, 61, 209, 1001)>,
<1168, 2, (209, 407)>,
<1221, 5, (5, 205, 606, 709, 807)>
]

... → ...

teaching → 258, [...]

Numerating the Words Inside a Document

include filename

```
grep -H -E -a -o '[A-Za-z]+' papers/1587.txt | tr 'A-Z' 'a-z' | cat -n  
1 papers/1587.txt:a  
2 papers/1587.txt:practical  
3 papers/1587.txt:approach  
4 papers/1587.txt:for  
5 papers/1587.txt:teaching  
6 papers/1587.txt:model  
7 papers/1587.txt:driven  
8 papers/1587.txt:software  
9 papers/1587.txt:development  
10 papers/1587.txt:a  
11 papers/1587.txt:plea  
12 papers/1587.txt:for  
...
```

numerize, starting
from 1

Remove Leading spaces, replace : with <tab>


```
$ grep -H -E -a -o '[A-Za-z]+' papers/1587.txt | tr 'A-Z' 'a-z' | cat -n | \
  sed 's#^ *##;s#:#\t#'
1      papers/1587.txt a
2      papers/1587.txt practical
3      papers/1587.txt approach
4      papers/1587.txt for
5      papers/1587.txt teaching
6      papers/1587.txt model
7      papers/1587.txt driven
8      papers/1587.txt software
9      papers/1587.txt development
10     papers/1587.txt a
11     papers/1587.txt plea
12     papers/1587.txt for
13     papers/1587.txt the
...
```

Loop Over Document Collection

```
rm -f position.Index
for f in papers/*.txt; do
    grep -H -E -a -o '[A-Za-z]+' $f | tr 'A-Z' 'a-z' | cat -n | \
        sed 's#^ *##;s#:#\t#' >> position.Index ;
done
less position.Index
1      papers/1580.txt gamification
2      papers/1580.txt technique
3      papers/1580.txt for
...
3465  papers/1580.txt page
1      papers/1581.txt traffic
2      papers/1581.txt lights
3      papers/1581.txt through
...
```


Sort by Term, File, Position

numeric sort



```
sort -k3 -k2 -k1n position.Index | tee sortedPosition.Index
84      papers/1580.txt a
88      papers/1580.txt a
103     papers/1580.txt a
139     papers/1580.txt a
...
3441   papers/1580.txt a
97      papers/1581.txt a
110     papers/1581.txt a
119     papers/1581.txt a
...
```

Distribution to Multiple Files

- Target Format:

```
$ ls invPosIndex/wo*.idx
invPosIndex/women.idx
invPosIndex/word.idx
invPosIndex/wordpress.idx
invPosIndex/work.idx
invPosIndex/workbench.idx
invPosIndex/worked.idx
...

$ cat invPosIndex/women.idx
papers/1582.txt 2274,2482,2505,2541,2975
papers/1588.txt 751,783,806,2480,2503,2539,2979,3338
...
```

awk-script

```
# Format : 208 papers/1151.txt about
# sort order: col3, col2 , col1
{
  if ($3 != last_term) {
    if (last_term != "")
      printf "\n" >> DIR"/invPosIndex/"last_term".idx"
    printf "%s\t%d", $2, $1 > DIR"/invPosIndex/"$3".idx"
    last_term = $3
    last_file = $2
  } else {
    if ($2!=last_file) {
      printf "\n%s\t%d", $2, $1 >> DIR"/invPosIndex/"$3".idx"
      last_file = $2
    } else {
      printf ",%d", $1 >> DIR"/invPosIndex/"$3".idx"
    }
  }
}
```

new term

end current line

new document (same term)

next position (same document & term)

Index Structure

```
$ head -15 invPosIndex/bachelor.idx      $ head invPosIndex/degree.idx
...
papers/1170.txt 1026,1513,1593,1841      papers/1173.txt 1747,1919
papers/1172.txt 1294                    papers/1176.txt 1813,5481,5597
papers/1189.txt 1946,1955,2614          papers/1189.txt 1397,1573,1947,1956,5234
papers/1190.txt 395                    papers/1190.txt 60,97,119,376,396,413,435, ...
papers/1191.txt 62,293,415,1637        papers/1191.txt 63,70,88,294,416,1638,2528
papers/1198.txt 95,704,1592            papers/1196.txt 2143
...
...
```

- Which documents contain the phrase „bachelor degree“
 - Both terms must appear
 - „bachelor“ must appear directly before „degree“

Index Structure

```

$ head -15 invPosIndex/bachelor.idx      $ head invPosIndex/degree.idx
...
papers/1170.txt 1026,1513,1593,1841      papers/1173.txt 1747,1919
papers/1172.txt 1294                    papers/1176.txt 1813,5481,5597
papers/1189.txt 1946,1955,2614          papers/1189.txt 1397,1573,1947,1956,5234
papers/1190.txt 395                    papers/1190.txt 60,97,119,376,396,413,435, ...
papers/1191.txt 62,293,415,1637        papers/1191.txt 63,70,88,294,416,1638,2528
papers/1198.txt 95,704,1592            papers/1196.txt 2143
...

```

- Which documents contain the phrase „bachelor degree“
 - Both terms must appear in a document
 - „bachelor“ must appear directly before „degree“

Query Execution

```
$ join -11 -21 $(DIR)/invPosIndex/bachelor.idx \
              $(DIR)/invPosIndex/degree.idx -o1.1,1.2,2.2 | \
awk -F' ' -f phraseMatch.awk | sort -k2,2nr
```

find documents which
contain both terms

look for cases, in which the term
„bachelor“ appears directly before „degree“

document

positionlist
for „bachelor“

positionlist
for „degree“

Example Query

- phraseMatch.awk

```

{
    num_matches = 0
    num_elem = split($0,a," ")
    num_fw = split(a[2], fw, ",")
    num_sw = split(a[3], sw, ",")
    i=1
    j=1
    do {
        if (fw[i]>sw[j]) {
            j++
        } else if (fw[i]+1==sw[j]) {
            i++
            j++
            num_matches++
        } else
            i++
    } while (i < num_fw && j < num_sw)
    if (num_matches > 0)
        print $1" "num_matches
}

```

positions at which „bachelor“ appear

positions at which „degree“ appear

here, we have a match

print number of matches for each document containing a match

Topics not Covered

- Stemming & Lemmatisation
- Vector Space Model
- Compression

Summary

- What was the purpose of this tutorial?
- Unix Tools like `grep`, `tr`, `sed`, `uniq`, `comm`, `sort`, `join` are very powerful tools for data scientists
- `awk` can be seen as a programming language with perfect fit to the previous mentioned tools
- Alternatives to `awk` are `python`, `perl`, `ruby`, `php`, ... (depends on own preferences)
- There are a number of other tools not covered in this tutorial like `paste`, `cut`, `zgrep`, `zcat`, `wget`, ... which are also worth to get discovered
- To glue all commands together, risk a look at `make`

*thanks for your audience
&
enjoy dbkda 2018*