Basic Corpus Processing

Sprachverarbeitung (VL + Ü)

Nils Reiter

April 11, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Recap: Command Line

- Login, entering commands
- Help and information about commands: `--help`, `man`
- File system navigation: `cd`, `ls`
  - Tab completion to automatically expand paths
- File system manipulation: `mkdir`, `rmdir`, `rm`, `cp`, `mv`
- Remote login: `ssh USERNAME@compute.spinfo.uni-koeln.de`

Today

- Input/output redirection, pipes
- File name patterns
- A few text tools for the command line
- Goal: Basic corpus processing: Counting words and measuring frequencies
- Don't overthink: We don't need this very often

Steps

1. Corpus identification
   - Which corpus exactly do we want to investigate?
   - In our case: Complete works of Edgar Allen Poe from Gutenberg dump
2. Preparations: Merge them into one file
3. Simple statistics
   - Count the words (= we're ignoring punctuation)
   - Get word frequencies

Section 1

1. Identify Works

Data

- In general: Depends on the data set and its organization
- Directory `/resources/gutenberg/` contains dump from project Gutenberg
- Huge number of files
- Each file corresponds to a book and has an integer id number
  - E.g.: id number 2149 can be found in directory `2/1/4/2149/`
- Ids are documented in index files
  - `GUTINDEX.00.txt`, `GUTINDEX-2011.txt`, ...

File Name Patterns

- File paths can contain wildcards
- `bla*.txt` matches on all file names that start with `bla` and end on `.txt`
  - `*` can be anything of any length
- `bla?.txt` matches on all file names of 8 characters length that start with `bla` and end on `.txt`
  - `?` can be any single character

File Name Patterns

- File paths can contain wildcards
- `bla*.txt` matches on all file names that start with `bla` and end on `.txt`
  - `*` can be anything of any length
- `bla?.txt` matches on all file names of 8 characters length that start with `bla` and end on `.txt`
  - `?` can be any single character

Why is that useful?

- Many commands accept multiple file names as arguments
- E.g. `cp /old-place/* /new-place/` copies all files from old-place to new-place

Command Line Tool: Grep

```
$ grep "some string" [FILE]
```

- Searches in file(s) – can be multiple files and/or patterns
- Relevant option: `-i` for case-insensitive search

To Do

1. Inspect one index file with `less` to learn about the file format
2. Search for the author across *all* index files using `grep` – identify collections of works
3. Note the numbers

Section 2

2. Merge them into one file

Input and Output Streams

- Each running program has by default three basic IO channels
  - (Programs may open additional channels to read from files etc.)
- Standard output (STDOUT): Regular, output of the program
  - In Java: `System.out.println("bla")` goes to the standard output
  - By default: The terminal
- Standard error output (STDERR): Reserved for error messages, no buffering
  - By default: The terminal
- Standard input (STDIN): Where the program reads from
  - By default: The keyboard

Input and Output Streams

Redirection

- `> FILE` Redirects STDOUT into FILE
  - ⚠ If FILE already exists, it will be overwritten
- `>> FILE` Redirects STDOUT into FILE, but appends at the end
- `2> FILE` Redirects STDERR into FILE, overwrites
- `< FILE` Read STDIN from FILE, and not from keyboard

Input and Output Streams

Redirection

- `> FILE` Redirects STDOUT into FILE
  - ⚠ If FILE already exists, it will be overwritten
- `>> FILE` Redirects STDOUT into FILE, but appends at the end
- `2> FILE` Redirects STDERR into FILE, overwrites
- `< FILE` Read STDIN from FILE, and not from keyboard
- `CMD1 | CMD2` redirects STDOUT from CMD1 into STDIN from CMD2
  - E.g. `$ grep -i poe GUTINDEX* | less`

Command Line Tool: Cat

```
$ cat [SOME_FILES]
```

- `cat` prints the entire content of the given files on the command line
- Output of `cat` can be redirected into a new file: `$ cat POE_FILES > Poe.txt`

To Do

- Generate a new file called Poe.txt that contains all the works by Poe

Section 3

3. Simple Statistics

Tokenization

Split the texts into tokens

- Today, we ignore punctuation
- General idea: Combination of tools `tr` and `wc`
- Intermediate goal: Each token on a separate line

Command Line Tool: Tr

```
$ tr SET1 SET2
```

- Translates strings: All occurrences of characters in SET1 are replaced by their counterparts from SET2
- Reads from standard input, writes to standard output

Command Line Tool: Wc

```
$ wc [SOME_FILE]
```

- Counts words, lines, characters and bytes in a file
- ⚠ Naive tokenization (i.e., by whitespace)
  - See for yourself: `$ echo "bla. blubb"| wc` detects two words
- Options can control to only count lines etc.

To Do

1. Use `tr` to make sure every word is on its own line
2. Use `wc` to count the lines

Word Frequencies

- Which words appear how often in a text?
- Get a list of all words, count each of them
- Basic idea: Use sort to group together the same words, then use uniq to collapse and count them
- Two additional commands: `sort` and `uniq`

Command Line Tool: Sort

```
$ sort [OPTIONS] [FILE]
```

- Sorts input according to various criteria
- Sorted result printed to STDOUT
- By default: Sort lines alphabetically
- Option `-n` ensures sorting numerically

Command Line Tool: Uniq

```
uniq [OPTIONS] [INPUT_FILE [OUTPUT_FILE]]
```

- Groups together subsequent equal lines
- Relevant option: `-c` count how many lines have been collapsed

Example

Original file:

```
apple
apple
peach
apple
```

Uniq'ed file:

```
apple
peach
apple
```

To Do

1. Use `tr` to make sure every word is on its own line
   - Re-use commands from before!
2. Sort alphabetically
3. Count how many rows can be collapsed
4. Sort numerically
5. Pipe into `less` to be able to read it

Section 4

Exercise

Exercise

Our project Gutenberg dump contains two editions of Doyles' »The Valley of Fear«. We want to study how they differ (if they differ).

- Find out their id numbers.
- Extract their word frequencies.
- Inspect and compare them (manually). Do you think it's the same text?