# Homework 1
# Text Normalization and Frequency Analysis

Swesan Pathmanathan
`pathmans@mcmaster.ca`

## 1 Data

I used two different text corpora to satisfy the $\geq$50,000-token requirement and to explore how the same processing pipeline behaves on very different types of text.

- **Crime and Punishment** by Fyodor Dostoyevsky, obtained from Project Gutenberg (ID 2554). The file is a plain-text version of a 19th-century English novel and contains 206,544 raw tokens. I chose this dataset because it is a well-known literary work written for human readers, with a large amount of dialogue and a rich vocabulary, making it a natural example of unstructured prose.
- **Linux system log excerpt** (Linux.txt), containing 308,039 raw tokens collected from authentication and kernel logs. The text consists of system messages, timestamps, service names, and status information. I chose this dataset because I am interested in automating debugging and monitoring workflows, and tokenizing log files is a common first step in structuring system events.

## 2 Methodology

### 2.1 Approach

I built the software incrementally, starting with a simple text-processing pipeline and expanding it as needed. The core of the program is a Python function that reads a plain-text file using UTF-8 encoding and performs manual tokenization by splitting on whitespace. I tested the pipeline first on the novel to make sure it handled standard prose correctly, and then ran it on the Linux log to ensure it could also process more technical text containing timestamps, identifiers, and system messages.

While working with the log file, I encountered a small number of malformed byte sequences caused by raw system and network input being written directly to the logs. To prevent decoding errors from stopping the program, I used UTF-8 decoding with error replacement so the rest of the file could still be processed. This allowed the pipeline to remain robust without affecting the overall frequency analysis.

After tokenization was working, I added normalization options one at a time. Lowercasing and stopword removal were implemented first, followed by stemming and lemmatization to handle different word forms. I added my custom option (`-myopt`) after noticing that both datasets contained many numeric tokens, such as dates, timestamps, and identifiers, which added noise without contributing meaningful lexical information.

To generate figures, I wrote the token frequency output to files and used matplotlib to create Zipf plots with logarithmic scales on both axes. I generated plots for both raw and normalized versions of each corpus to see how different normalization steps affected the frequency distribution.

### 2.2 Normalization Options

The program supports the following command-line options, which can be enabled in any combination:

- **-lowercase**: converts all tokens to lowercase to reduce vocabulary size.
- **-stopwords**: removes common English stopwords using the NLTK stopword list.
- **-stem**: applies Porter stemming to reduce words to their root forms.
- **-lemmatize**: applies WordNet lemmatization to normalize words to their dictionary form (this option cannot be used together with `-stem`).
- **-myopt**: removes tokens that consist only of digits (e.g., dates, timestamps, and process IDs). I added this option because numeric tokens appeared frequently in both datasets and increased vocabulary sparsity without improving the interpretability of word frequency results.

### 2.3 Tokenization

Tokenization is implemented using a manual regular-expression-based split on whitespace rather than external tokenizer libraries, in order to satisfy the assignment constraints. Punctuation remains attached to tokens unless affected by a normalization step, keeping the tokenization process simple and explicit.

## 3 Sample Output

### 3.1 Sample Output (Crime and Punishment, 206,544 raw tokens)

**Top 25 (most frequent, raw).**

| Token | Count | Token | Count |
|---|---|---|---|
| the | 7404 | it | 1751 |
| and | 6053 | with | 1698 |
| to | 5189 | not | 1638 |
| a | 4433 | had | 1555 |
| of | 3813 | for | 1514 |
| I | 3424 | from | 1445 |
| he | 3364 | her | 1413 |
| in | 2985 | on | 1316 |
| was | 2737 | is | 1259 |
| you | 2734 | she | 1164 |
| that | 2529 | He | 1139 |
| his | 1985 | as | 1129 |
| at | 1926 | | |

**Bottom 25 (least frequent, raw).**

| Token | Count | Token | Count |
|---|---|---|---|
| donate | 1 | visit | 1 |
| 5 | 1 | professor | 1 |
| michael | 1 | hart | 1 |
| originator | 1 | library | 1 |
| network | 1 | volunteer | 1 |
| support | 1 | editions | 1 |
| included | 1 | thus | 1 |
| necessarily | 1 | edition | 1 |
| pg | 1 | facility | 1 |
| includes | 1 | gutenberg | 1 |
| ebooks | 1 | subscribe | 1 |
| email | 1 | newsletter | 1 |
| addresses | 1 | | |

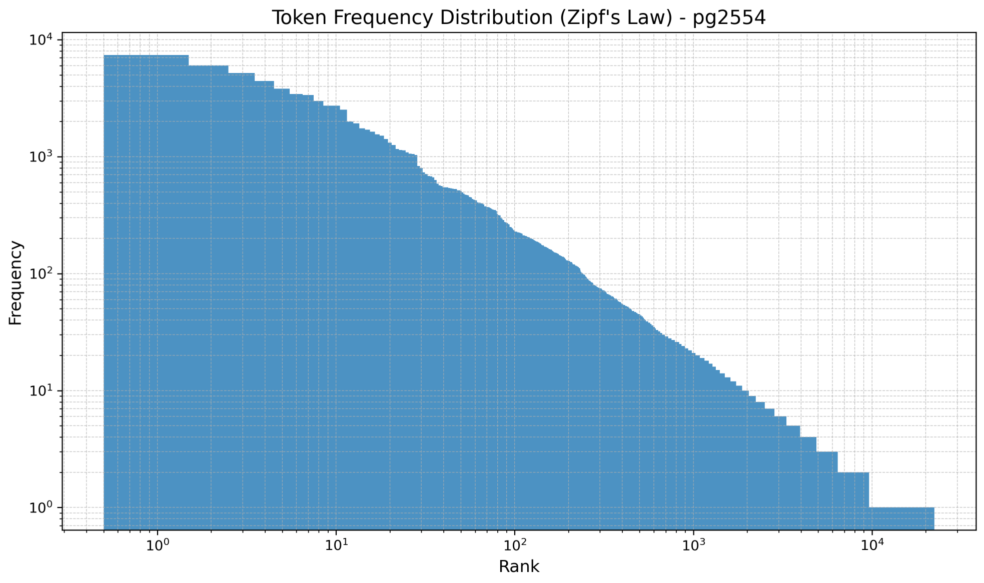

Figure 1: Zipf plot for Crime and Punishment (raw tokens).

### 3.2 Sample Output (Crime and Punishment, normalized 107,251 tokens)

Normalized using `-lowercase -myopt -stopwords -lemmatize`.

**Top 25 (most frequent, normalized).**

| Token | Count | Token | Count |
|---|---|---|---|
| would | 566 | man | 327 |
| raskolnikov | 546 | him | 316 |
| one | 533 | even | 314 |
| could | 476 | must | 294 |
| dont́ | 417 | looked | 288 |
| like | 414 | see | 286 |
| know | 408 | go | 284 |
| though | 399 | time | 274 |
| said | 399 | thatś | 268 |
| i | 392 | it | 266 |
| come | 387 | him | 256 |
| itś | 378 | little | 255 |
| went | 349 | | |

**Bottom 25 (least frequent, normalized).**

| Token | Count | Token | Count |
|---|---|---|---|
| addresses. | 1 | checks, | 1 |
| donations. | 1 | donate, | 1 |
| visit: | 1 | michael | 1 |
| hart | 1 | originator | 1 |
| library | 1 | network | 1 |
| support. | 1 | editions, | 1 |
| included. | 1 | thus, | 1 |
| necessarily | 1 | edition. | 1 |
| pg | 1 | facility: | 1 |
| includes | 1 | gutenberg™, | 1 |
| ebooks, | 1 | subscribe | 1 |
| newsletter | 1 | ebooks. | 1 |
| volunteer | 1 | | |

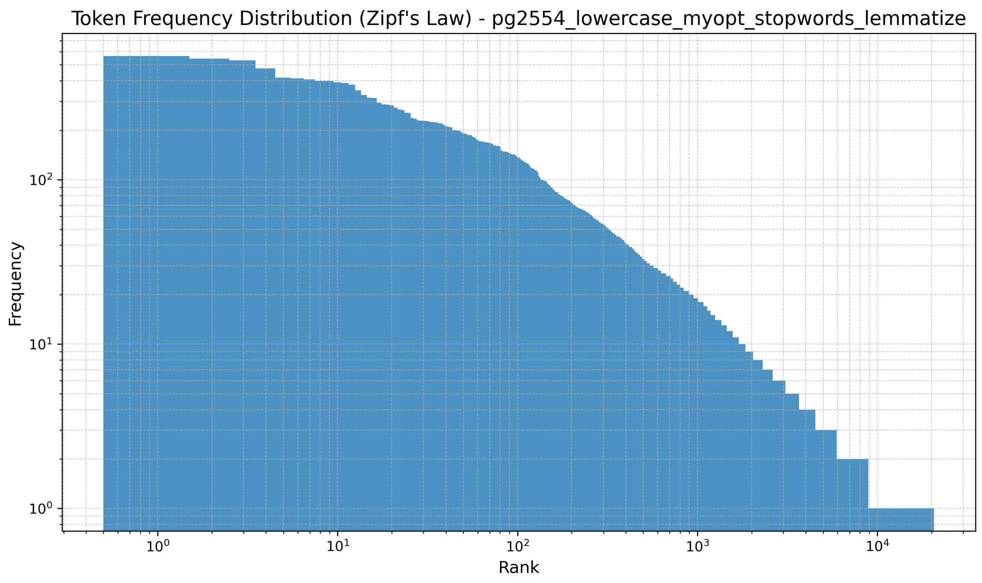

Figure 2: Zipf plot for Crime and Punishment (normalized with lowercase, digit removal, stopword removal, and lemmatization).

Linux log outputs (raw and normalized) exceeded 300k and 274k tokens respectively, with dominant tokens such as `combo` and `kernel:` characteristic of system-level events.

## 3.3 Sample Output (Linux system log, 308,039 raw tokens)

**Top 25 (most frequent, raw).**

| Token | Count | Token | Count |
|---|---|---|---|
| combo | 25567 | ruser= | 3940 |
| kernel: | 13670 | logname= | 3935 |
| process | 10452 | Jan | 2829 |
| of | 10428 | 2005 | 2597 |
| Memory: | 10412 | Jul | 2407 |
| Out | 10404 | user=root | 2258 |
| Killed | 10404 | Sep | 2204 |
| Nov | 9744 | user | 2177 |
| (httpd). | 8903 | 21 | 2028 |
| Dec | 4816 | Jun | 1885 |
| euid=0 | 4049 | Oct | 1815 |
| uid=0 | 4047 | 22 | 1699 |
| authentication | 3965 | | |

**Bottom 25 (least frequent, raw).**

| Token | Count | Token | Count |
|---|---|---|---|
| ftpd[6664]: | 1 | ftpd[6665]: | 1 |
| 06:45:42 | 1 | sshd(pam_unix)[6666]: | 1 |
| 06:45:48 | 1 | sshd(pam_unix)[6668]: | 1 |
| unix_chkpwd[6678]: | 1 | sshd(pam_unix)[6670]: | 1 |
| unix_chkpwd[6679]: | 1 | sshd(pam_unix)[6671]: | 1 |
| unix_chkpwd[6680]: | 1 | sshd(pam_unix)[6675]: | 1 |
| unix_chkpwd[6681]: | 1 | sshd(pam_unix)[6674]: | 1 |
| 01:48:59 | 1 | sshd(pam_unix)[6721]: | 1 |
| sshd(pam_unix)[6728]: | 1 | sshd(pam_unix)[6726]: | 1 |
| sshd(pam_unix)[6724]: | 1 | sshd(pam_unix)[6723]: | 1 |
| sshd(pam_unix)[6727]: | 1 | sshd(pam_unix)[6725]: | 1 |
| sshd(pam_unix)[6736]: | 1 | 04:48:54 | 1 |
| sshd(pam_unix)[6741]: | 1 | | |

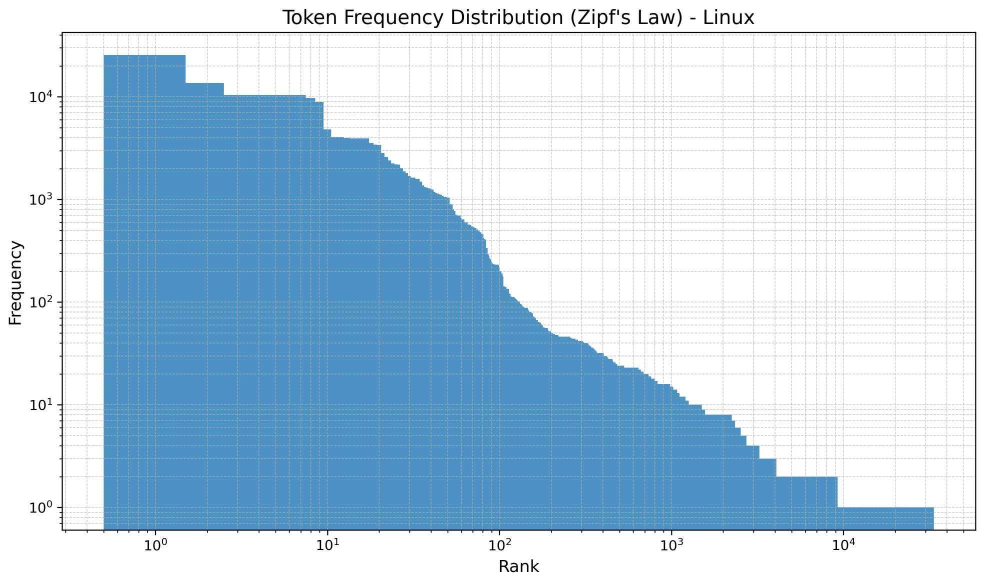

Figure 3: Zipf plot for Linux system log (raw tokens).

## 3.4 Sample Output (Linux system log, normalized 232,980 tokens)

Normalized using `-lowercase -myopt -stopwords -stem`.

**Top 25 (most frequent, normalized).**

| Token | Count | Token | Count |
|---|---|---|---|
| combo | 25567 | tty=nodevssh | 3934 |
| kernel: | 13670 | failure; | 3928 |
| process | 10452 | connect | 3516 |
| memory: | 10436 | host | 3421 |
| kill | 10432 | jan | 2829 |
| nov | 9744 | jul | 2407 |
| (httpd). | 8903 | user=root | 2258 |
| dec | 4816 | user | 2213 |
| euid=0 | 4049 | sep | 2204 |
| uid=0 | 4047 | jun | 1885 |
| authent | 4008 | oct | 1815 |
| ruser= | 3940 | aug | 1637 |
| logname= | 3935 | | |

**Bottom 25 (least frequent, normalized).**

| Token | Count | Token | Count |
|---|---|---|---|
| ftpd[6664]: | 1 | ftpd[6665]: | 1 |
| 06:45:42 | 1 | sshd(pam_unix)[6666]: | 1 |
| 06:45:48 | 1 | sshd(pam_unix)[6668]: | 1 |
| unix_chkpwd[6678]: | 1 | sshd(pam_unix)[6670]: | 1 |
| unix_chkpwd[6679]: | 1 | sshd(pam_unix)[6671]: | 1 |
| unix_chkpwd[6680]: | 1 | sshd(pam_unix)[6675]: | 1 |
| unix_chkpwd[6681]: | 1 | sshd(pam_unix)[6674]: | 1 |
| 01:48:59 | 1 | sshd(pam_unix)[6721]: | 1 |
| sshd(pam_unix)[6728]: | 1 | sshd(pam_unix)[6726]: | 1 |
| sshd(pam_unix)[6724]: | 1 | sshd(pam_unix)[6723]: | 1 |
| sshd(pam_unix)[6727]: | 1 | sshd(pam_unix)[6725]: | 1 |
| sshd(pam_unix)[6736]: | 1 | 04:48:54 | 1 |
| sshd(pam_unix)[6741]: | 1 | | |

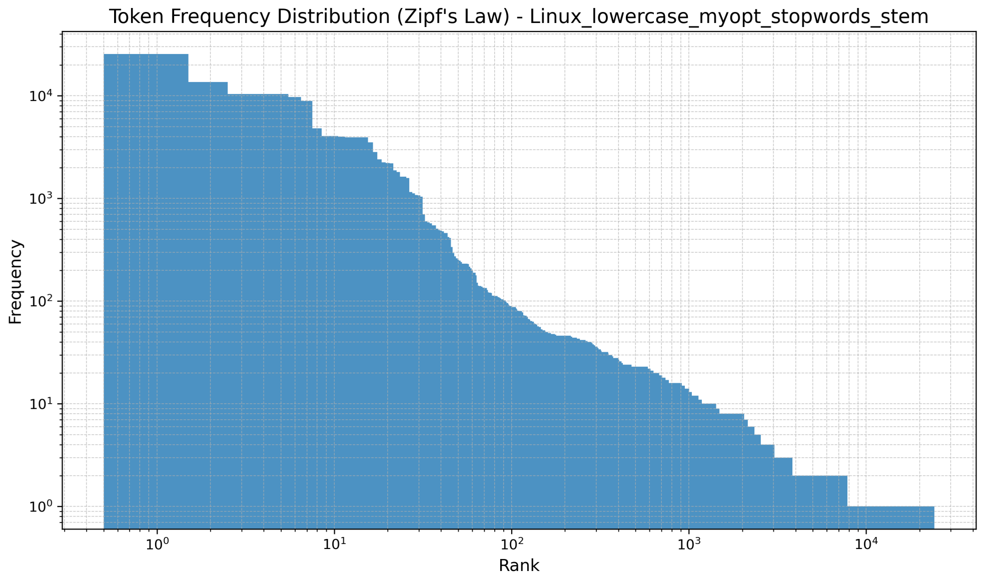

Figure 4: Zipf plot for Linux system log (normalized with lowercase, digit removal, stopword removal, and stemming).

## 4 Discussion

### 4.1 Observations on Token Frequency and Zipf's Law

When I looked at the most frequent tokens in my results, clear patterns emerged immediately. In the novel, the highest-ranked tokens were almost entirely function words such as "the", "and", "to", "a", and "of". These words appear frequently because they are structurally necessary for English sentences rather than because they carry much semantic meaning. In contrast, the Linux log was dominated by repeated system labels and metadata such as "combo", "kernel:", and "process". Although these are not natural language words, they play a similar structural role in logs by organizing and labeling events. After removing stopwords, the most frequent tokens in the novel shifted toward content words like "raskolnikov" and "would", which better reflect the subject matter of the text.

At the opposite end of the frequency lists, I observed a long tail of tokens that appeared only once, consistent with Zipf's Law. In the novel, these were mostly rare vocabulary items, proper nouns, and residual Project Gutenberg metadata. In the Linux log, most single-occurrence tokens were timestamps, process IDs, and one-off service messages. This effect was especially strong in the log corpus, where the majority of unique token types appeared only once, illustrating how real system data naturally produces a very long tail of low-frequency tokens.

### 4.2 Comparison to Wikipedia on Zipf's Law

I compared my results to the Wikipedia discussion of word frequencies in natural languages. Both corpora exhibited the characteristic steep drop-off in frequency from the most common tokens and a long tail of rare tokens, matching the general Zipfian pattern described there. However, the Linux log showed a more linear relationship on the log–log plot than the novel. This is likely because logs repeat a small, fixed vocabulary of event labels very consistently, whereas novels use a broader and more flexible range of vocabulary.

These differences highlight that while Zipf's Law captures a general trend, the exact shape of the distribution depends heavily on the domain. The novel reflects linguistic variety, while the log reflects repetitive system behavior.

### 4.3 Impact of Stopword and Digit Removal

Removing stopwords had a large impact on the total number of tokens, reducing the novel from 206,544 tokens to 107,251 tokens (about a 48% decrease). Despite this large reduction, the remaining vocabulary still followed a Zipfian shape. This happens because stopwords occur extremely frequently but contribute little semantic information. By contrast, removing digit-only tokens had a much smaller effect on the total token count because most numeric tokens appear only once or a few times.

Filtering out digit-only tokens reduced vocabulary sparsity by removing many single-occurrence items such as dates, timestamps, version numbers, and process IDs. This pruned the long tail of the distribution and made the remaining token frequencies more representative of actual lexical patterns. While this removes numeric information, it is a reasonable trade-off for an assignment focused on word frequency analysis rather than numerical semantics.

### 4.4 What I Learned

This assignment exposed several practical challenges involved in working with real-world text. I initially assumed that NLTK resources such as stopwords and WordNet would be available by default, but I learned that these corpora must be downloaded explicitly. Handling these cases programmatically made the tool more robust and user-friendly. I also gained experience structuring a command-line tool with argparse and managing UTF-8 file I/O in a way that does not fail on imperfect input.

Working with the Linux log reinforced how strongly domain affects preprocessing choices. Normalization steps that make sense for prose can have very different effects on technical text. Building the pipeline iteratively—starting with a simple tokenizer, inspecting the output, and then adding targeted normalization steps—proved far more effective than trying to design an ideal solution upfront. Overall, this project helped make the trade-offs involved in text normalization much more concrete.

## Generative AI Usage Disclosure

Generative AI tools (ChatGPT) were used to assist with code structuring, debugging, and drafting the report. All final content was reviewed and edited by the author.

**Carbon Footprint Estimate** An approximate carbon footprint was estimated using the published value of 4.32 g $CO_2$ per ChatGPT query. Approximately 25 queries were used, resulting in an estimated total of 108 g $CO_2$. This estimate is approximate and likely an underestimate due to limited transparency regarding model hardware and deployment region.