STAN49 - Assignment 1

William Nordansjö

February 20, 2025

Introduction

In this assignment, I will investigate two laws related to textual data analysis: The first one is Heaps' law which concerns the relationship between the size of a text document and the size of its corresponding vocabulary and the other way around. The second law is Zipf's law which states that the frequency of a word is inversely proportional to its position in the list of most common words. Finally I will attempt to build a classification model which aims to find the identity of the author of a book based on documents of book components. The report is thus divided into these tasks, where task 1 and 2 will share the same dataset and build upon each other while task 3 is a standalone section.

Task 1: Heaps' law

This section will investigate Heaps' law as mentioned in the introduction. More formally, the law states a relation between a size of a text document ($|D|$) and a size of a vocabulary ($|V|$) which corresponds to it. This can be expressed mathematically as in equation (1),

$$|V| = a|D|^b \tag{1}$$

where $a, b > 0$. This can intuitively be understood that the growth of the vocabulary will slow down even though more words are considered if $1 > b > 0$, or conversely that the growth increases if $b > 1$.

This supposed relationship will be investigated by examining 50 randomly selected documents from a dataset of news articles from the BBC. The dataset consists of 2225 documents relating to five topics

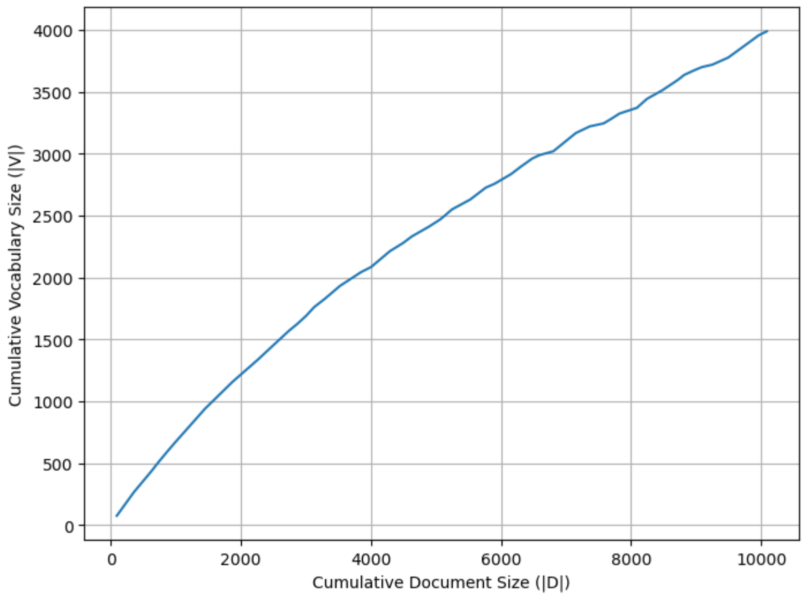

Figure 1: Heaps' Law: Vocabulary Growth vs. Document Size

(business, entertainment, politics, sports and tech) published on the BBC news website between the years $2004 - 2005$ (Greene & Cunningham 2006). The working dataset used in this assignment was thus created by randomly sampling 50 of these documents and subjecting them to some data pre-processing. First the words were coerced to be lowercase, dots and commas were removed along with english stopwords such as "the", "is" and "and". Following this, the cumulative vocabulary and document sizes were calculated and plotted against each other which can be seen in figure 1.

Figure 1 is thus a visual representation of Heaps' law and we can clearly see how the relationship takes shape, where new unique words starts to occur less frequently even though more text is introduced. This implies that we have a $b < 1$, in other words the rate of change in $|V|$ is decreasing as $|D|$ decreases however all the while $|V|$ is growing. This makes sense if you consider a theoretical scenario: Imagine discovering the library of a long lost civilization with their own language, one would expect that as more and more of the language is *uncovered* the rate of new words *discovered* would be expected to decrease as the more common words would presumably be logged quickly. This effect is apparent in the BBC case even when removing stopwords which per definition are very common.

Before estimating the parameters, we can do a log-log transformation in order to get a linear relationship. Equation 2 shows the final form from which we can estimate the parameters using the least squares as $\hat{a}$: 2.461, $\hat{b}$: 0.807. The linear can be seen in figure 2 and again shows a diminishing relationship. An interesting point of discussion whether or not the order affects the relationship, it might have an initial

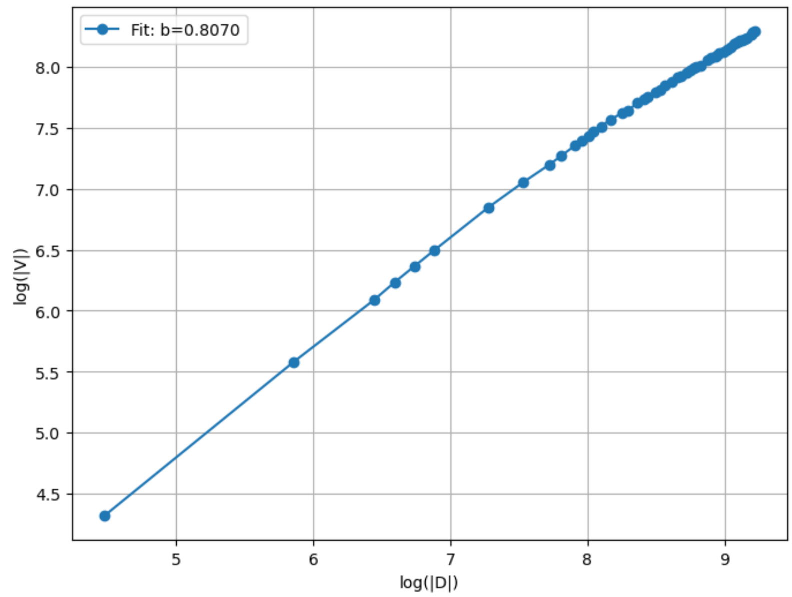

Figure 2: Log-Log Plot of Vocabulary Growth

effect if starting with the largest documents but the main mechanism of diminishing vocabulary growth should remain as it is reasonable to presume that each document should on average has the same ratio of unique to non-unique words and as such the composition of any randomly chosen documents would be more or less similar.

$$log(|V|) = log(a|D|^b)$$

$$log(|V|) = log(a) + log(|D|^b)$$

$$log(|V|) = log(a) + b \cdot log(|D|) \tag{2}$$

Task 2: Zipf's law

This section will cover Zipf's law by further exploring the BBC dataset as above. This law relates to term frequency and rank (in terms of frequency) and states that

$$f_i = \frac{c}{i} \tag{3}$$

where $c > 0$. So what is f_i? It is the frequency of the word t_i, where t_1 is the most frequent word and t_{1+1} is the second most frequent word. This means that the frequency of a word supposedly is inversely proportional to the rank of the word.

In order to investigate this relationship, each word was counted and assigned a rank based on frequency as shown in table 1. Similar to the previous section this law can be expressed as a linear relationship by taking the logarithm. First we rewrite equation 3 as

$$f_i = c \cdot i^{-1} \tag{4}$$

and taking the logarithm on both sides gives again

$$log(f_i) = log(c) - 1 \cdot log(i). \tag{5}$$

where we have a constant $log(c)$ and an negative slope of $log(i)$ where i is corresponds to rank.

Table 1: Word Frequencies and Ranks

Word	**Frequency**	**Rank**
said	175	1
also	64	2
people	63	3
would	53	4
mr	52	5

Figure 3 shows the log of frequency on the y-axis and the log of rank on the x-axis, it visualizes the empirical relationship given by the BBC dataset as the blue line and the fitted line as the red. Using the least squares method again the estimated parameters were found as $\hat{c}$: 379.434, and slope or $\hat{b}$: -0.725. The slope in this scenario should ideally be -1, as seen in equation 5, in order to describe the inverse

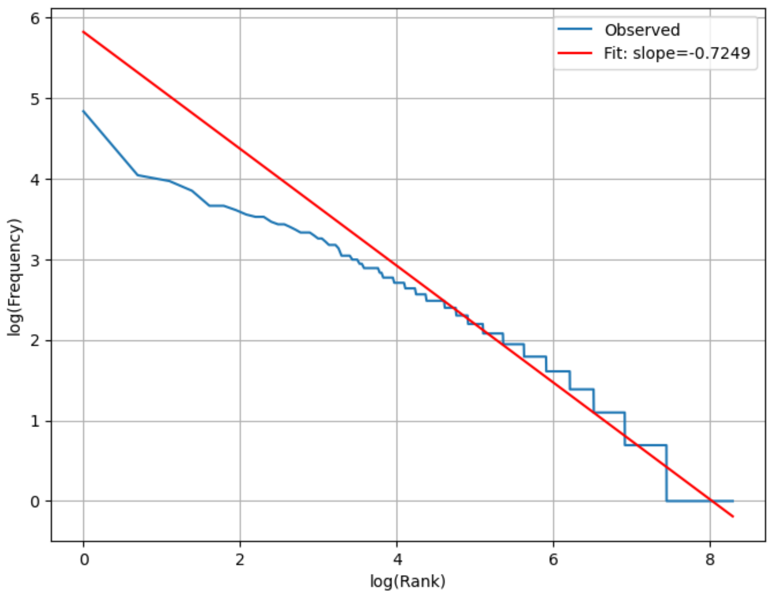

Figure 3: Log-Log Plot of Zipf's Law

proportional relationship explicitly, which would give the rank 1 word a frequency of ≈ 379. Instead as table 1 shows our observed frequency for our rank 1 word, *said* was 175, resulting in a difference of 204. This does not necessarily disprove Zipf's law, a possible reason for this difference could be the stopword removal step of the pre-processing which excluded some of the most frequent words in the dataset.

Task 3: Identify the author

In the final section of this assignment, a classifier was built in order to attempt to discern between the authors of two books. The chosen books are the novels The Trial by Franz Kafka and Mrs. Dalloway by Virginia Woolf, the reasoning behind the choices is that they both are contemporaries of each other, with the publishing date of the two books being published in 1925. Other than that the two books are very distinct with authors of a different sex and different countries, Woolf being from England and Kafka from what was then the Austrian-Hungarian Empire or today's Czech Republic. Mrs. Dalloway was also written in english from the start while The Trial was translated. They are also different in disposition with Kafka favoring longer, rather winding paragraphs and Woolf her more stunted stream of consciousness style.

Practically, the two books were loaded, split up into documents by paragraphs and labeled by author either as Woolf or Kafka. After this the documents (paragraphs) were split into training and testing datasets, balanced so that an equal number of documents were considered and finally shuffled. Once in this familiar format they were further processed, first by converting them to tf-idf vectors and as such giving weight to more uncommon words in the documents. Following this. a similar pre-processing to the one used in task 1 and 2 was performed, removing dots, commas and stopwords. Once fully prepared, a regularized logistic regression model was trained. Logistic regression was chosen as it is an extremely simple model that generally achieves good results, it was cross-validated over an elastic net, allowing the model to try different ratios of L_1 and L_2 regularization. This approach allows the data to guide the model specification rather than strictly either LASSO or Ridge beforehand.

The accuracy of the classifier was 0.9, which implies that the model was very successful in predicting which author wrote the paragraph considered. More detailed metrics of classification performance can be found in table 2, where the model scored high in terms of F_1 as well as in both precision and recall, with a slightly higher precision in predicting Kafka and vice versa.

The reason for this high performance is likely three-fold: First of all how distinct the novels were, despite being published in the same year and being written in english, the Central European naming conventions of the translated Kafka novel and the place names of the very english Mrs Dalloway is sure to provide a clear hint to the model. Secondly the choices made while processing the text, such as dividing documents upon paragraphs and converting the data into tf-idf vectors, further enhancing the language differences. Finally the elastic-net logistic regression model proved powerful enough to pick up upon the tendencies of the data while remaining general enough to classify the unseen data accurately.

Table 2: Classification Performance Metrics

Class	**Precision**	**Recall**	**F1-score**	**Support**
Kafka	0.92	0.88	0.90	41
Woolf	0.88	0.93	0.90	41
Macro avg	0.90	0.90	0.90	82
Weighted avg	0.90	0.90	0.90	82

Conclusion

This report has investigated two laws of textual data analysis and created a text-based classification model. In task 1 Heaps' law was successfully demonstrated using 50 randomly selected BBC news documents, showing a clear decrease of unique words added to the vocabulary as new documents were introduced. In task 2 Zipf's law was investigated, and while not as clearly demonstrated, evidence of an inversely proportional relationship between term frequency and rank of frequency was found although with a slope of ≈ -0.7 instead of -1. Finally two books of different authors written on the same year was chosen. These were split into documents and labeled, and a classification model was trained in order to predict the author given a document from either of the books. The classification model achieved a very high accuracy of 90% when considered the processed data and being tuned using elastic net regularization.

References

D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

Appendix A: Python Code

```
import numpy as np
import pandas as pd
import os
import random
import re
import pandas as pd
from collections import Counter
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from IPython.display import display
import matplotlib.pyplot as plt
from scipy.stats import linregress
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import accuracy_score, classification_report


random.seed(704)  # For reproducibility

def preprocess_text_1(text):
    text = text.lower()
    text = re.sub(r"[^\w\s]", "", text)  # Remove punctuation
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stopwords.words("english")]  #
        Remove stopwords
    return tokens
```

```python

base_dir = "bbc"

#Finding the texts
text_files = []
for category in os.listdir(base_dir):
    category_path = os.path.join(base_dir, category)
    if os.path.isdir(category_path):
        for filename in os.listdir(category_path):
            if filename.endswith(".txt"):
                text_files.append(os.path.join(category_path, filename))

num_samples = min(50, len(text_files))
random_files = random.sample(text_files, num_samples)

document_texts = []
for file_path in random_files:
    with open(file_path, "r", encoding="utf-8") as f:
        document_texts.append(preprocess_text_1(f.read()))

cumulative_sizes = []
cumulative_vocab_sizes = []
vocab_set = set()
total_words = 0

for doc in document_texts:
    total_words += len(doc)  # Adding document size to total
    vocab_set.update(doc)  # Adding new words to vocabulary

    cumulative_sizes.append(total_words)
    cumulative_vocab_sizes.append(len(vocab_set))

df = pd.DataFrame({
    "Cumulative_Doc_Size": cumulative_sizes,
    "Cumulative_Vocab_Size": cumulative_vocab_sizes
})


```

```python
plt.figure(figsize=(8,6))
plt.plot(df["Cumulative_Doc_Size"], df["Cumulative_Vocab_Size"], linestyle='-')
plt.xlabel("Cumulative Document Size (|D|)")
plt.ylabel("Cumulative Vocabulary Size (|V|)")
plt.grid(True)
plt.show()

log_D = np.log(df["Cumulative_Doc_Size"])
log_V = np.log(df["Cumulative_Vocab_Size"])

slope, intercept, r_value, _, _ = linregress(log_D, log_V)

a = np.exp(intercept)
b = slope

print(f"Estimated a: {a:.4f}")
print(f"Estimated b: {b:.4f}")

plt.figure(figsize=(8,6))
plt.plot(log_D, log_V, marker='o', linestyle='-', label=f"Fit: b={b:.4f}")
plt.xlabel("log(|D|)")
plt.ylabel("log(|V|)")
plt.legend()
plt.grid(True)
plt.show()

all_words = [word for doc in document_texts for word in doc]
word_freq = Counter(all_words)
sorted_word_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)

zipf_df = pd.DataFrame(sorted_word_freq, columns=["Word", "Frequency"])
zipf_df["Rank"] = zipf_df.index + 1  # Rank starts from 1

log_rank = np.log(zipf_df["Rank"])
log_freq = np.log(zipf_df["Frequency"])

slope, intercept, r_value, _, _ = linregress(log_rank, log_freq)
c = np.exp(intercept)  # Since log c = intercept
```

```python

print(f"Estimated c: {c:.4f}")
print(f"Estimated slope (should be ~ -1): {slope:.4f}")

# Plot log-log relationship with fitted line
plt.figure(figsize=(8,6))
plt.plot(log_rank, log_freq, '-', label="Observed")
plt.plot(log_rank, slope * log_rank + intercept, 'r-', label=f"Fit: slope={slope:.4f}")
plt.xlabel("log(Rank)")
plt.ylabel("log(Frequency)")
plt.legend()
plt.grid(True)
plt.show()

with open('pg71865.txt', encoding='utf-8') as f:
    woolf_paragraphs = [p.strip() for p in f.read().split("\n\n") if p.strip()]

with open('pg7849.txt', encoding='utf-8') as f:
    kafka_paragraphs = [p.strip() for p in f.read().split("\n\n") if p.strip()]

woolf_docs = [(p, "Woolf") for p in woolf_paragraphs]
kafka_docs = [(p, "Kafka") for p in kafka_paragraphs]

split_woolf = int(0.8 * len(woolf_docs))
split_kafka = int(0.8 * len(kafka_docs))

woolf_train, woolf_test = woolf_docs[:split_woolf], woolf_docs[split_woolf:]
kafka_train, kafka_test = kafka_docs[:split_kafka], kafka_docs[split_kafka:]

#For balance
train_size = min(len(woolf_train), len(kafka_train))
test_size = min(len(woolf_test), len(kafka_test))

train_data = woolf_train[:train_size] + kafka_train[:train_size]
test_data = woolf_test[:test_size] + kafka_test[:test_size]

random.shuffle(train_data)
```

```
random.shuffle(test_data)

train_df = pd.DataFrame(train_data, columns=["Text", "Author"])
test_df = pd.DataFrame(test_data, columns=["Text", "Author"])

test_df.head()

# Modified preprocessing function
def preprocess_text_2(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stopwords.words('english')]
    return " ".join(tokens)

train_df["Processed_Text"] = train_df["Text"].apply(preprocess_text_2)
test_df["Processed_Text"] = test_df["Text"].apply(preprocess_text_2)


tfidf_vectorizer = TfidfVectorizer(max_features=5000)

X_train = tfidf_vectorizer.fit_transform(train_df["Processed_Text"])
y_train = train_df["Author"]
X_test = tfidf_vectorizer.transform(test_df["Processed_Text"])
y_test = test_df["Author"]

# Classifier with elastic net reg
classifier = LogisticRegressionCV(
    penalty="elasticnet",
    solver="saga",
    l1_ratios=[0.1, 0.5, 0.9],
    cv=5,
    max_iter=1000
)

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```python
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(report)
```

Appendix B: Generative AI usage

ChatGPT was used to brainstorm and to help me develop my ideas for approaching the tasks, also as an alternative to google for debugging code.