Final Project - Report

Sandra Poernbacher - Statistical Methods for Language Processing

In the Final Project discussed topics in the context of Natural Language Processing (NLP) are applied on a literary text from the Gutenberg library. All the computations are done in R programming language and involve the analysis of bigram patterns, testing statistical relationships, the measurement of linguistic randomness through entropy, and more. The goal of the Project is to demonstrate how computational tools can uncover patterns and insights in large textual dataset while applying statistical methods.

1 Choosing a Text

I started the project by using the `gutenbergr` package in R to search and download a suitable text for analysis. The criteria for selection were that the authors last name should be closest to my last name, and the text must contain more than 100.000 words. To identify an author whose last name was closest to mine, I filtered the authors whose last names started with “Por” using the `unique` and `startsWith` functions. After browsing through the works of the displayed authors, I selected a text by Jane Porter, which also met the second criterion (the book contains ~286.000 words). The chosen work is titled “The Scottish Chiefs”, available under Gutenberg ID 6068. I downloaded the text using the `gutenberg_download()` function, specifying a mirror site for efficient access. This text was then prepared for further analysis in the following steps.

```
library(gutenbergr)
library(tidytext)
library(dplyr)
library(stringr)

my_mirror <- "http://mirrors.xmission.com/gutenberg/"

df <- gutenberg_metadata

unique(df$author)[startsWith(unique(df$author), "Por")]

gutenberg_works(author == "Porter, Jane")

ScotChi <- gutenberg_download(6086, mirror = my_mirror)
```

In addition to the `gutenbergr` library, I used other essential libraries in this project to improve text processing and analysis. The `tidytext` library was used for tokenizing the text into smaller units such as unigrams and bigrams, enabling further frequency analysis and other NLP tasks. The `dplyr` library facilitated data manipulation and summarization, such as filtering, grouping, and counting words. Finally, the `stringr` library provided tools for string manipulation that were useful in handling and preprocessing textual data.

2 Bigram Count

In the second step of the project, I performed a word and bigram count to analyze the structure and descriptive features of the text. The first step involved tokenizing the text into individual words with the `unnest_tokens()` function, creating a dataset where each row represented a single word. I then computed the word (unigram) frequencies using the `count()` function on `words_ScotChi`, sorting the results in descending order by frequency. The words with the highest frequencies like "the", "of" or "to" can often be classified as stopwords or functional words (e.g. prepositions, conjunctions). Out of 12.748 tokens in total 4.455 appeared only once in the whole data.

After tokenizing the unigrams, I continued with bigrams by specifying the `token = "ngrams"` parameter with `n=2`. The frequency calculation of bigrams was done similar to the unigrams. I added a filter to clear any rows with missing or invalid data: `[!is.na(count_bigrams$words), ]` checks if the value of a `words` is set to `NA`. The resulting bigram counts revealed commonly co-occurring word pairs, such as "of the", "to the" and "in the". Out of 11.583 bigrams in total 81.910 are unique.

Although unigrams and bigrams with high frequency are often disregarded as they may include stopwords, other high-frequency n-grams with multiple tokens can uncover common phrases or expressions that reflect the text's style or content. These counts can serve as a basis for further analysis, such as identifying key topics or distinguishing authors.

```
words_ScotChi <- unnest_tokens(ScotChi, words, text)

count_unigrams <- count(words_ScotChi, words, sort=TRUE)

bigrams_ScotChi <- unnest_tokens(ScotChi, words, text, token = "ngrams", n=2)

count_bigrams <- count(bigrams_ScotChi, words, sort=TRUE)

count_bigrams <- count_bigrams[!is.na(count_bigrams$words), ]
```

3 Chi-Square Test & Contingency Table

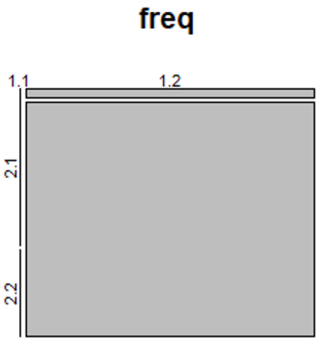
Step 3 includes a deeper look at the dependence between the words in a selected bigram using a chi-square test. For this purpose the bigram "going to" was chosen as an example of a typical expression. Its frequency is compared against other bigrams containing "going" or "to" or neither of them. The contingency table represents the co-occurrence frequencies of the words "going" (w_1) and "to" (w_2):

	w_1	not w_1
w_2	27	15
not w_2	8922	252368

Contingency Table

The contingency table shows that the counts for the co-occurrence of w_1 and w_2 ("going to"), the occurrence of w_1 without w_2, the occurence of w_2 without w_1, and the absence of both w_1 and w_2 . These counts allow a comparison of observed frequencies with expected frequencies under the null hypothesis of independence.

After computing the different frequencies, a matrix `freq` is created that represents the contingency table. It includes the frequencies discussed in the previous paragraph. The parameter `ncol = 2` specifies that the matrix is made of two columns, and `byrow = TRUE` ensures that the values are filled row by row and not in another order. The `mosaicplot(freq)` generates a mosaic plot for the contingency table to represent the proportional frequencies visually. The following `chisq.test(freq)` performs a chi-square test on the contingency table. The matrix `contingency_table` is similar to `freq` but it includes row and columns names as.



Mosaicplot for Contingency Table

```
count_bigrams[startsWith(count_bigrams$words, "going to"),]
count_bigrams[startsWith(count_bigrams$words, "going "),]
count_bigrams[endsWith(count_bigrams$words, " to"),]

g.t <- count_bigrams[startsWith(count_bigrams$words, "going to"),]$n
g.nott <- sum(count_bigrams[startsWith(count_bigrams$words, "going "),]$n) -
notg.t <- sum(count_bigrams[endsWith(count_bigrams$words, " to"),]$n) - g.t
notg.nott <- sum(count_bigrams$n) - g.nott -notg.t - g.t

freq <- matrix(c(g.t, g.nott, notg.t, notg.nott), ncol = 2, byrow = TRUE)
mosaicplot(freq)
chisq.test(freq)

# Contingency Table
contingency_table <- matrix(
  c(g.t, g.nott, notg.t, notg.nott),
  ncol = 2,
  byrow = TRUE,
  dimnames = list(c("w2", "not w2"), c("w1", "not w1"))
)

contingency_table
mosaicplot(contingency_table, main="Mosaic Plot of Bigram Dependence", color
```

The chi-square test determines whether the words “going” w_1 and “to” w_2 in the selected bigram “going to” occur independently within the text. By constructing a contingency table of the co-

occurrences and non-occurrences of the words “going” and “to”, the test compared the observed frequencies with the expected frequencies under the assumption of independence. The resulting chi-square statistic was 452.27 with 1 degree of freedom, the p-value was less than 2.2e-16, which is far below a typical significance level (close to zero). This confirms a significant dependence between the words “going” and “to” and suggests they frequently occur as a fixed phrase in the text. However, the chi-square approximation may be inaccurate, as indicated by the warning:

```
Warning:
In chisq.test(freq) : Chi-squared approximation may be incorrect
```

The warning message during the test indicates that some expected frequencies in the contingency table were too small to satisfy the assumptions of the chi-square test. This can occur when the sample size is too small or certain categories have low counts.

4 Hypothesis Test & Chi-square Test

A hypothesis test is a statistical method used to decide whether a given hypothesis about a population parameter is supported by the data. It starts with the formulation of two hypotheses: the zero hypothesis (H_0) and the alternative hypothesis (H_1). The zero hypothesis typically represents the assumption of no effect or no difference, while the alternative hypothesis suggests the presence of an effect or difference. The process involves calculating a test statistic from the sample data, which measures how much the observed data deviate from what is expected under H_0. This test statistic is then compared to a critical value coming from the sampling distribution. Or its corresponding p-value is calculated, which quantifies the probability of observing the test statistic or something more extreme if H_0 is true. If the p-value is smaller than a predetermined significance level (α, typically close to zero for example: 0.05), H_0 is rejected in favor of H_1, suggesting that the observed effect is statistically significant.

A chi-square test is a specific type of hypothesis test used to assess relationships between categorical variables or to test goodness-of-fit. It is based on the chi-square statistic: $X^2 = \sum_{j=1}^{m} \frac{(N_j - n_{0j})^2}{n_{oj}}$. In the context of this project, the chi-square test was applied to evaluate the independence of words in a bigram. The observed and expected frequencies were derived from the contingency table, and the test assessed whether the co-occurrence of words was significantly different from what would be expected under independence. This provides insight into whether the words in the bigram are dependent or occur together by chance.

5 Entropies of 1000 word strings

In step five, I calculated the entropy for each 1,000-word segment of the text to analyze the variability and unpredictability in the linguistic structure. Entropy measures the randomness or uncertainty in a given segment, and higher entropy values indicate greater diversity or less predictability in character usage.

First, an empty vector `entropy` is initialized. The text contains around 286,000 words, so I divided it into 285 segments, each containing 1,000 words. The loop limit was set to 285 to avoid going out of bounds, as the last segment starts at 285.000, leaving the remaining words in the 286th segment. I iterated through the text using a for loop, where each iteration processed one segment. In the loop, I extracted the words for the current segment and tokenized them into individual characters using the `unnest_tokens()` function. The results are converted into a dataframe `df.char`, that includes

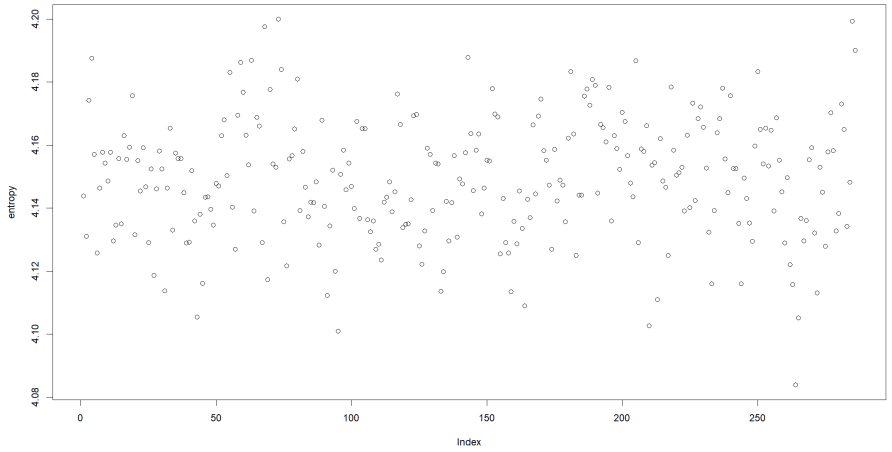
columns for each token and its frequency. For each character, I computed its relative frequency and used the entropy formula: $H = -\sum_i p_i log(q_i)$. In this case, p_i is the relative frequency of each character. After computing all segments, the previously initialized `entropy` vector contains all entropy values for the 1000 word parts and is visualized with `plot(entropy)`.

```
entropy <- c()

for(i in 0:285)
{
  entr <- words_ScotChi[(i*1000 + 1):(i*1000+1000), 2]
  char <- unnest_tokens(entr, token, words, token = "characters")
  df.char <- as.data.frame(count(char, token, sort = TRUE))
  df.char$relfreq <- df.char$n/sum(df.char$n)
  df.char$sent <- df.char$relfreq*log2(df.char$relfreq)
  entropy <- c(entropy, - sum(df.char$sent))
}

entropy
plot(entropy)
```

The computed entropy values varied across the 285 segments, revealing differences in the predictability of character distributions in different parts of the text. Segments with higher entropy values suggest greater linguistic variability, likely corresponding to more complex or diverse passages. Conversely, lower entropy values indicate more repetitive or predictable sections of the text. The plot of entropy provides a visual representation of these variations, highlighting how the text's structure and complexity shift over its length. This analysis sheds light on the text's stylistic and structural patterns.



6 Entropy of a Single Variable

Entropy in information theory is a measure of the informational value of a message. The key assumption is that the more likely a message is, the less information it conveys. For example, learning that a specific lottery number does not win provides almost no information since the probability of a number not winning is very high. On the other hand, discovering that a specific number wins conveys a significant amount of information because the probability of a number

winning is very low. Therefore, information can be viewed as a measure of uncertainty or surprise. Mathematically, the entropy H of a random variable $X$is defined as: $H(X) = E(-log_2(P(X)) = -\sum_x p(x)log_2p(x)$.

Entropy reaches its maximum when all outcomes are equally likely, reflecting the highest uncertainty. Conversely, it is zero when one outcome has a probability of 1 (complete certainty). In practical terms, entropy can be used to measure the diversity or complexity of a system. For example, in text analysis, higher entropy indicates more variation and unpredictability in character or word distributions, while lower entropy suggests repetition or predictability. Entropy serves as a fundamental concept for understanding randomness, variability, and information content in data.

7 The 95% Confidence Interval

In step 7, I calculated the 95% confidence interval for the entropies of the text. A confidence interval provides a range of values within which the true mean entropy of the population is likely to fall with a specified level of confidence—in this case, 95%. This calculation was based on the mean `mean_entropy` and standard `sd_entropy` deviation of the computed entropy values across all 1,000-word segments and used the t-distribution due to the finite sample size. To compute the t-distribution, the critical value `t_crit` is computed. Finally, the print function is used to display the calculated limits of the 95% confidence interval by adding a comma between the values `","`.

```
# Calculate the mean and standard deviation of entropy
mean_entropy <- mean(entropy)
sd_entropy <- sd(entropy)
n <- length(entropy)

# Compute the 95% confidence interval using the t-distribution
alpha <- 0.05
t_crit <- qt(1 - alpha/2, df = n - 1)

margin_of_error <- t_crit * (sd_entropy / sqrt(n))
lower_bound <- mean_entropy - margin_of_error
upper_bound <- mean_entropy + margin_of_error

print("95% Confidence Interval for Entropy: [", lower_bound, ",", upper_bound
```

The output of step 7 shows the 95% confidence interval for the mean entropy of the text, which is calculated as $[4.146975, 4.151441]$ $[4.146975, 4.151441]$. This interval suggests that, based on the sample of entropy values from the text, we are 95% confident that the true mean entropy of the entire text lies within this range. The small width of the interval indicates a high level of precision in the estimate, reflecting relatively low variability in the entropy values across the text segments. This finding suggests that the text maintains a fairly consistent level of randomness or complexity in its character distributions throughout. Overall, these results show that the text is written in a consistent style with a balanced use of vocabulary.

8 The Confidence Interval

A confidence interval is a range of values used to estimate an unknown population parameter, such as a mean, based on sample data. It provides a level of confidence, usually expressed as a percentage (e.g., 95%), that the true value of the parameter lies within this range. The width of the interval depends on the variability in the data and the sample size; a smaller variability or a larger sample size results in a narrower interval, indicating greater precision. For example, a 95% confidence interval means that if the same sampling process were repeated many times, 95% of the intervals calculated would contain the true population parameter.

9 Naive Bayes Classifier

In step 9, I divided the text into four approximately equal sections and used a Naive Bayes classifier to determine which section a given sentence most likely belongs to. The selected sentence was extracted from line 147 of the text, split into individual words, and treated as the input for the classification task.

First, I divided the entire text into four sections with `ceiling(n/4)`, each containing an equal number of rows. This ensured a balanced dataset for comparison. For each section, I calculated the probability of each word appearing in that section. These probabilities were then used to compute the likelihood of the selected sentence belonging to each section by iterating over the section with `lapply()`. Using the Naive Bayes principle, the probability of the sentence being in a section was calculated as the product of the probabilities of its words occurring in that section. The `sapply()` function iterates over the list of word probabilities and filters for each section the word present in the example sentence. Finally, these words are multiplied using `prod()` while `na.rm = TRUE` ensures that missing probabilities are ignored to not cause errors. The last section prints the the probability for each section and the index of the section with the highest probability: `(which.max(naive_bayes_probs))`.

```
# Extract the example sentence
example_sentence <- ScotChi$text[147] # Line 147
example_sentence

# Split into sentences to get one whole sentence starting in this line
example_sentence <- str_split(example_sentence, "\\.\\s+", simplify = TRUE)[1
example_sentence

# Adjust Naive Bayes classification to use this sentence
# Divide text into 4 sections
n <- nrow(words_ScotChi)
section_size <- ceiling(n / 4)
sections <- list(
  section1 = words_ScotChi[1:section_size, ],
  section2 = words_ScotChi[(section_size + 1):(2 * section_size), ],
  section3 = words_ScotChi[(2 * section_size + 1):(3 * section_size), ],
  section4 = words_ScotChi[(3 * section_size + 1):n, ]
)

# Calculate word probabilities for each section
word_probs <- lapply(sections, function(section) {
  section %>% count(words) %>% mutate(prob = n / sum(n))
```

```
})

# Compute the Naive Bayes probability for the sentence in each section
naive_bayes_probs <- sapply(word_probs, function(wp) {
  probs <- wp %>% filter(words %in% sentence) %>% pull(prob)
  prod(probs, na.rm = TRUE) # Multiply probabilities
})

# Normalize to get relative probabilities
naive_bayes_probs <- naive_bayes_probs / sum(naive_bayes_probs)

# Output the classification
print("Sentence classification probabilities:\n")
print(naive_bayes_probs)
print("Classified as section:", which.max(naive_bayes_probs), "\n")
```

The Naive Bayes classifier returned probabilities for each of the four sections, representing the likelihood of the sentence belonging to each section. The section with the highest probability was chosen as the classification result. For example, if section 3 had the highest probability, it suggests that the sentence's word distribution is most similar to the word probabilities in section 3. This approach demonstrates the utility of Naive Bayes in text classification tasks, even with the simplifying assumption that words occur independently within a section

10 Conclusion

This project applied various natural language processing (NLP) techniques to analyze a literary text from the Gutenberg library, highlighting the value of computational tools in exploring large textual datasets. Key findings include the identification of frequent unigrams and n-grams, which revealed common patterns and stylistic features of the text. The chi-square test demonstrated significant dependencies in bigrams like "going to," providing insights into fixed expressions within the text. Entropy calculations illustrated the consistency of linguistic randomness, reflecting a uniform style throughout the text. Additionally, the Naive Bayes classifier classified sentences into sections based on word probabilities, showing the power of probabilistic models in text classification.

Overall, this analysis not only offered a deeper understanding of the text's structure and complexity but also demonstrated how statistical and computational methods can uncover meaningful patterns in language. These techniques are broadly applicable across various domains, from literary analysis to machine learning and beyond, highlighting the importance of integrating statistical tools into NLP workflows.