Faculty of
Mathematics
and Informatics

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MASTER'S STUDY PROGRAMME
MODELLING AND DATA ANALYSIS

Lithuanian Text Difficulty Characterization by Syllables Frequencies

Lietuviškų tekstų sudėtingumo analizė

Master's Thesis

Author: Laima Štulaitė

Supervisor: Marijus Radavičius, Prof., Dr. (HP)

Vilnius
2024

Abstract

The frequency of words in a language is well-described by Zipf's (1949) law. However, studies at the syllable level are relatively rare in the field of quantitative linguistics, and Zipf's law does not necessarily describe the distribution of syllables. In examining the frequency of syllable occurrence in the Lithuanian language, I found that the ranked frequencies of syllables are best described by the Yule distribution model. The Yule equation fits the distribution of Lithuanian syllable rank frequencies better than the Zipf's, Beta, and Zipf-Mandelbrot models. To account for the complexity of the Lithuanian language, I employed Shannon and conditional entropy measures. The Shannon entropy rate averaged 8.91 information bits per syllable across the Lithuanian text corpus, and the conditional entropy averaged 6.45, conditioned on the preceding syllable. The Shannon entropy rate was used to classify more complex texts, and the gradient boost classification algorithm demonstrated the best accuracy and balance in classifying fractions of syllables from 80 Lithuanian texts into complex and not complex categories.

Keywords: Zipf’s law; Yule model; Beta model; Zipf-Mandelbrot; rank-frequency distribution; syllable‘s entropy rate; syllable’s conditional entropy; complex text classification; gradient boost classification.

Santrauka

Žodžių dažnumą kalboje gerai apibūdina Zipfo (1949) dėsnis. Tačiau kiekybinės lingvistikos srityje skiemenų lygmens tyrimai yra palyginti reti, o Zipfo dėsnis nebūtinai apibūdina žodžius sudarančių skiemenų pasiskirstymą. Tirdama skiemenų pasiskirstymo dažnumą lietuvių kalboje, nustačiau, kad skiemenų ranginius dažnius geriausiai apibūdina “Yule” pasiskirstymo modelis. “Yule” lygtis lietuvių kalbos skiemenų ranginių dažnių pasiskirstymą atitinka geriau nei Zipfo, Beta ir Zipfo-Mandelbroto modeliai. Siekdama įvertinti lietuvių kalbos sudėtingumą, taikiau Šenono ir sąlyginės entropijos matus. Šenono entropijos rodiklis vidutiniškai siekė 8,91 informacijos bito visame lietuviškų tekstų korpuse, o sąlyginė entropija - 6,45, priklausomai nuo prieš tai esančio skiemens. Šenono entropijos matas yra naudojamas sudėtingesniems tekstams klasifikuoti, o gradientinis stiprinimo klasifikavimo algoritmas parodė geriausią tikslumą ir pusiausvyrą klasifikuojant 80 lietuviškų tekstų skiemenų dalis į sudėtingų ir nesudėtingų tekstų kategorijas.

Raktažodžiai: Zipfo dėsnis; “Yule” modelis; Beta modelis; Zipfo-Mandelbrotas; rangų dažnių pasiskirstymas; skiemenų entropijos norma; skiemenų sąlyginė entropija; sudėtingų tekstų klasifikavimas; gradientinis stiprinimo klasifikavimas.

Content

Introduction

The aim of this thesis is to explore difficulty of the Lithuanian language through the lenses of syllable frequencies, information entropy and syllabic bigram networks. The study examines a syllabified corpus of 80 Lithuanian texts, including both original and translated works, to offer an understanding of the syllables as linguistic structure. Central to the linguistic exploration is the frequency of linguistic units such as words, syllables, and phonemes. Several found studies [1],[2] have been engaged in applying Zipf's law rank-frequency distribution commonly used mostly for words *(Eq. 1, 2*) in linguistic analyses to the syllables. This thesis aligns with these studies, seeking to understand whether the methods used for word frequency analysis can also effectively describe the rank-frequency relation of Lithuanian syllables.

In addition to syllable frequency analysis, the study employs Shannon entropy and conditional entropy as tools to quantify linguistic difficulty. These measures, recognized in the field of linguistics [4], are used to estimate the information content inherent in syllables. Following the methodology used by Pellegrino et al. (2007), this study uses Shannon entropy to quantify the average information derived from unigram syllables probability distributions, thereby assessing the difficulty of the Lithuanian language based on the information rate of syllables.

Another aspect of this research is the analysis of syllabic bigram networks. This involves mapping the connections and frequencies of syllable pairings within the text corpus, providing insights into the syntactic structures of the Lithuanian language. The bigram network analysis shows patterns and tendencies in syllable usage, revealing structural elements that are not immediately apparent through syllable's unigram analysis alone. These analytical tasks were performed in the R program.

Finally, the thesis incorporates a classification of complex text. Parts of texts labelled as 'Complex' and 'Not Complex' were derived from Shannon entropy measure. For classification gradient boosting, logistic regression, random forest, decision tree and k-nearest neighbours (KNN) classification algorithms were employed. Classification exercise was implemented in Python.

Literature review

Ranked frequency distribution studies

Research on the level of syllables appears relatively rare. One can assume that it might be due to lack of precise and unified definitions of syllables and methods of syllabification. Radojičić et al. (2019) made an analysis of several properties of syllables, one of them being the rank-frequency distribution. The idea behind in that part is to check already formulated hypothesis by Strauss et al. (2008) that the rank-frequency distribution of syllables behaves like the rank-frequency distribution of words. Zipf-Mandelbrot model is a generalization of Zipf's law that introduces a new parameter b. Zipf-Mandelbrot that model was used for the application:

$$p_r = \frac{C}{(r+b)^a} \qquad Eq.1$$

The distribution has two free parameters a ($a > 0$) and b $[0, \infty)$, C is a scaling constant. Radojičić et al. (2019) present results of supporting the hypothesis of Strauss et al. (2008), that syllables behave like words and mimic word behaviour in terms of rank - frequency. When investigating several syllable properties Rujević et al. (2021) followed the research of Wimmer & Altmann (1999) and already reviewed Radojičić et al. (2019) on syllable's rank-frequency characterization and applied the same The Zipf-Mandelbrot (Eq. 1) model to the rank-frequencies of syllables of ten chapters from a Russian novel and its translations into Croatian, Serbian, and Ukrainian. The Zipf-Mandelbrot distribution is applied to the rank-frequencies of syllables and achieves a good fit. This model is more often used for word frequencies and as well as in many other scientific disciplines. Both authors support the hypothesis from Strauss et al. (2008), in which the rank-frequency distribution of syllables can be modelled by the same model for words, but also concludes that the parameter values for syllables differ quite dramatically from those for words.

Li at al. (2010) discuss the need to employ two-parameter functions over the traditional one-parameter model, such as Zipf's law [11], for fitting ranked linguistic data for several motives. One of them is that ranked linguistic data like phoneme's frequency or other, do not follow very well Zipf's power law:

$$p_r = \frac{C}{r^a} \qquad Eq.2$$

here p_r represents the probability of the r-th most frequent item, C is a constant, and a is the scaling exponent that determines how steeply the frequency declines with rank - r. While a is a free parameter constrained to be positive $(a > 0)$, C serves as a scaling factor to adjust the distribution. When Zipf's

law is represented on a logarithmic scale, the law is linearized, and the constant C translates to the y-intercept. In this linearized form, the y-intercept is not always functionally significant, as it may be absorbed into the normalization of the probability distribution or adjusted during regression analysis.

Authors suggest that it is worth using models with two parameters in data fitting for more flexibility. Other reason, even in the well-known case of Zipf's Law of ranked-frequency distribution the fitting often is not that good when the full rank-frequency range is investigated. This imperfection is ignored rather than analysed. Authors are eager to see if a two-parameter functions can be used to model the rank-frequency linguistic data better than Zipf's law. One of the considered is a Beta function that attempts to approximate the two sides low and high ranks of the rank-frequency distributions. The frequencies, denoted as $f_{(1)} \geq f_{(2)} \geq \cdots \geq f_{(n)}$, are arranged in descending order, where $f_{(1)}$ is the highest frequency, $f_{(2)}$ is the second highest, and so on, down to $f_{(n)}$. These frequencies are used to calculate the normalized frequencies, represented as $p_{(r)} \equiv f_{(r)}/N$, so that $\sum_{r=1}^{n} p_{(r)} = 1$, ($n$ is the number of items to be ranked, r is rank, and normalization factor $N = \sum_{r=1}^{n} f_{(r)}$. In the Beta function, the rank distribution is modelled as:

$$p_r = \mathrm{C}\frac{(\mathrm{n} + 1 - \mathrm{r})^b}{r^a} \qquad Eq.3$$

where the parameter a, which must be greater than zero ($a > 0$), characterizes the scaling of the distribution at lower ranks, affecting the most frequent items and b ($b \in [0, \infty)$) characterizes the scaling for the high-rank-number (low frequency) items points. The total number of syllable types is denoted by n, and C is a normalization constant that is adjusted based on the values of a, b and n to ensure that the sum of the distribution equals 1.

The second two-parameter function that was applied is the Yule function:

$$p_r = \mathrm{C}\frac{\mathrm{b}^r}{r^a} \qquad Eq.4$$

It has two parameters, a ($a > 0$) and b ($b \in [0, \infty)$), which influences the distribution's curvature, C is a scaling constant. Yule function has been used in several studies of linguistics units [5], [8], mostly to describe phonemes.

Li at al. (2010) discussed Zipf-Mandelbrot *(Eq. 1)* as another two-parameter function, proposed by Mandelbrot [18], but since it cannot be easily cast in a regression framework it was not used in their paper.

Finally, authors conclude that both Beta function and Yule function are fitting better than Zipf's law, and that the two-parameter model is worth using even though many empirical linguists prefer functions

with one key parameter like Zipf law. The intention is to find a balance between the poor fitting with too few parameters and overfitting with too many parameters.

Shannon entropy

In relatively recent years, information theory, particularly the concept of Shannon entropy, has become a fundamental tool in communication and linguistic research. It is quite commonly used principle which allows us to quantify the information in bits. The concept of Shannon entropy has been considered and used as a quantitative measure of difficulty in linguistics [12]. Developed by Claude Shannon, this concept quantifies the information content in messages, measured in bits, and provides a mathematical framework for analysing communication channels between a source and a receiver. Shannon's entropy $H_L\ Eq.9$ measures the unpredictability or randomness of information content, essentially quantifying the average amount of information produced by a stochastic source of data. This concept is crucial in determining the theoretical limits of effective communication in a channel. If the information transmitted is below the channel capacity, it can result in ambiguity where messages become indistinguishable. Conversely, transmitting information above the channel's capacity leads to redundancy and inefficiency [12], [14].

Extending beyond Shannon entropy, conditional entropy $\big(H(X|C)\big)\ Eq.10$ offers further insights, particularly in linguistic complexity. This measure quantifies the amount of information in a linguistic unit (such as a syllable or a word) given the knowledge of another unit. It's especially useful in understanding the dependencies and structure within language.

The application of information-theoretic measures like Shannon entropy $(H_L)\ Eq.9$ and conditional entropy $\big(H(X|C)\big)\ Eq.10$ offers a method to quantify linguistic complexity. These measures can be particularly insightful when analysing language as a series of messages between a speaker and a listener. Within this framework, a language or text can be considered difficult if it contains a high amount of information concerning its linguistic units, such as syllables.

Methodology

Syllable definition

A syllable, intuitively felt in speech, often eludes precise definition, especially in spoken language where distinct boundaries between syllables are not always apparent. Various theories attempt to define syllable boundaries, ranging from those based on articulatory features (such as muscle tension in the speech apparatus and the volume of air exhaled) to acoustic or auditory features (like spectral analysis and consonant voicing). The challenge in establishing objective syllable boundaries based on phonetic features underscores the notion that the number of syllables in a language may be more significant than their precise demarcations. In this context, a syllabication tool for computer processing of language was developed by G. Norkevičius, A. Kazlauskienė, G. Raškinis, A. Vaičiūnas, and A. Petrovas, as described in [13], and is available online or for downloading.

Description of the data

The dataset for this thesis comprises texts recommended for pupils in Years 5-8, drawn from a collection of Lithuanian and foreign authors. These works are available in the digital library http://ebiblioteka.mkp.emokykla.lt/. The collection includes 80 texts from 63 different authors, spanning to 8 genres. The total word count of the texts was initially 215,717. Following data pre-processing and cleaning, which involved removing some syllables that were not correctly syllabified (along with the corresponding words), the dataset was refined to 199,669 unique words. The original syllabified version contained approximately 10,800 different syllables; after correction, this number was reduced to 8,109 unique syllables. The texts were processed into 12 syllables groups using the Automatic Syllabication Software developed in [13].

Data Pre-processing

The data pre-processing stage was mostly guided by the syllable structure of the Lithuanian language as described in Kasparaitis P.'s work [16]. The syllable structure is shown in the formula STRARTSK, representing the fundamental building blocks of Lithuanian syllables:

S represents: {'s', 'š', 'z', 'ž'}.

T represents: {'b', 'd', 'g', 'k', 'p', 't', 'c', 'č', 'dz', 'dž', 'ch', 'h', 'f'}.

R represents: {'j', 'l', 'm', 'n', 'r', 'v'}.

A stands for any vowel or group of vowels, an essential component in every syllable.

K represents: {'k', 't'}.

In this structure, the presence of an element from set A (vowel or vowel group) is mandatory, underlining its significance as the core of a syllable. Other elements (S, T, R, and K) are optional.

We applied the STRARTSK formula for text segmentation into syllables, which was instrumental in identifying incorrect syllable constructions. This process involved pinpointing the vowel groups (A) labelled as V (vowel) and then placing them in context with preceding or succeeding consonant groups (C) represented by S, T, R, and K. To validate the accuracy of syllabification, we classified syllables into types based on their constituent phonemic elements, adhering to the constraints specified in the Lithuanian syllabic structure STRASTK.

Syllabic Bigram Analysis

In our investigation into the structure of the Lithuanian language, we employed a first-order Markov chain model. This approach is rooted in the principles of bigram analysis, commonly utilized in natural language processing. A Markov chain, in this analysis context, shows transitions from one syllable to another, with the probability of each state transition depending only on the current state and not the events that precede it.

$$P(S_{n+1} = s_j \mid S_n = s_i) = p_{ij} \qquad Eq.5$$

In this formula, S_{n+1} denotes the syllable following the current syllable S_n, and p_{ij} represents the probability of transitioning from syllable s_i to s_j. These probabilities are derived from the frequency of each syllable pair in the dataset:

$$p_{ij} = \frac{\text{Frequency of bigram } (s_i, s_j)}{\text{Total occurrences of syllable } s_i} \qquad Eq.5.1$$

The decision to use bigrams pairs comes from our finding that the average number of syllables per word in analysed Lithuanian text corpus is approximately 2.19.

Syllable's rank-frequency distribution

Ranking syllables by their frequency may establish the priorities in learning a Lithuanian language and prioritize learning most common syllables as well as help understand syllabification better. Fitting a power-law function for the whole range of ranks is by no mean perfect. In several studies it was suggested that power-law fitting with two different scaling exponents are needed to fit the ranked syllables frequencies.

To check more closely how well Zipf's law (*Eq. 2*) and other models the Beta (*Eq. 3*), Yule (*Eq. 4*) fit our syllable frequency data which have been used in several quotative linguistic analyses, I conducted multiple regression analyses. These analyses included a standard multiple regression of all rank-frequency points for syllables. To enhance the robustness of our models, especially in the presence of heteroscedasticity or non-normal distribution of residuals, I also implemented weighted regression models. Weighting scheme:w_1 implements a monotonically decreasing weight of $1/r$, where r is the rank of the syllable. This weighting approach assigns higher importance to syllables with lower ranks (more frequent syllables).

Each syllable occurs in the text with some frequency. Frequency here means the number of occurrences in this text that it appears. A syllable has a frequency f and frequency rank r. Both r and f are numbers, we can think of the relationship between them as a function. I want to see if and if so, how well Lithuanian syllables rank-frequency relation can be defined not only by already explored methods for syllables, but also by introduced methods for word and other linguistic unit's frequencies.

The frequencies denoted as f are ranked $f\_((1)) \geq f\_((2)) \geq \cdots \geq f\_((n))$ then normalized frequency $p_{(r)} \equiv f_{(r)}/N$, so that $\sum_{r=1}^{n} p_{(r)} = 1$, ($n$ is the number of items to be ranked, r is rank, and normalization factor $N = \sum_{r=1}^{n} f_{(r)}$. One of the first considered is a Beta function that attempts to approximate the rank-frequency distributions in *Eq. 3*. Applying a logarithmic transformation to both sides of Beta function (*Eq. 3*) transforms it into a multiple regression model:

$$\log(p_r) = c_0 + c_1 \log(r) + c_2 \log(n + 1 - r) \qquad Eq.3.1$$

In this form, $c_0 = \log(C), c_1 = -a$, and $c_2 = b$.

The second two-parameter function that was applied is the Yule function which is expressed in *Eq. 4*. Which both sides can be logarithmically transformed into the multiple regression model as well:

$$\log(p_r) = c_0 + c_1 \mathrm{r} + c_2 \log(r) \qquad Eq.4.1$$

In this form $c_0 = \log(C), c_2 = -a$, and $c_1 = \log(b)$.

I am also interested to see how well Zipf's law in *Eq. 2* can fit the data. Both sides can as well be logarithmically transformed into a regression model:

$$\log(p_r) = c_0 + c_1 \log(r) \qquad Eq.2.1$$

In this form $c_0 = \log(C)$, $c_1 = -a$.

Besides these multiple regression models, I also explore the Zipf-Mandelbrot model to understand its applicability to Lithuanian syllables' rank-frequency distribution. The Zipf-Mandelbrot model is a generalization of the Zipf model and is expressed in *Eq. 1*. The inclusion of the Zipf-Mandelbrot

model allows us to assess whether the adjustment of ranks with parameter b provides a more accurate representation of the syllable frequency distribution in Lithuanian as Radojičić et al. (2019) experimented with Serbian language syllables. To estimate the parameters of this model, a non-linear least squares method is employed. This approach is particularly suited for the Zipf-Mandelbrot model because of its inherent non-linearity, which is a characteristic that sets it apart from the more straightforward linear relationships found in the Zipf, Beta, and Yule models after their respective logarithmic transformations.

Above functions are variations of the power law function, so the model is better fitted for logarithmic frequency scales which was done in *Eq. 2.1, 3.1, 4.1*. This will also allow determining if one of these distributions has a chance of describing ranked syllable frequencies better than the other. In regression analysis, the model $y = F(x)$ parameters are estimated by minimizing the sum of squared errors:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - F(x_i))^2 \qquad Eq.6$$

Models can be compared using Akaike information criterion:

$$AIC = n \log \frac{SSE}{n} + 2K \qquad Eq.7$$

where K is the number of fitting parameters in the model.

For the weighted models, the Sum of Squared Errors (SSE) is calculated by taking into account w_1 weight assigned to each data point. The formula for weighted SSE is:

$$SSE_w = \sum_{i=1}^{n} (w_{1i}(y_i - F(x_i))^2) \qquad Eq.6.1$$

Therefore, Akaike information criterion for weighted models is:

$$AIC = n \log \frac{SSE_w}{n} + 2K \qquad Eq.7.1$$

In the process of model comparison, it is important to note that the methodology for comparing AIC values is applicable to both standard and weighted models. This uniform approach in comparing AIC values, regardless of the model type, enables consistent evaluation of model performance. For two models applied to the same dataset, their AIC differences are:

$$\Delta AIC = AIC_2 - AIC_1 = n \log \frac{SSE_2}{SSE_1} + 2(K_2 - K_1) \qquad Eq.8$$

Where AIC_1 and AIC_2 are the AIC values for two different models, SSE_1 and SSE_2 are their respective Sum of Squared Errors, K_1 and K_2 represent the number of parameters in each model. A smaller AIC value indicates a better model.

Information-theoretic complexity measures

In text data analysis, understanding the information content is essential. Text is comprised of symbols, most commonly letters, each occurring with varying frequencies. To digitally represent text, we often use ASCII (American Standard Code for Information Interchange), where each character is assigned a number from 0 to 127, occupying 7 bits (Picture 2 [15]). For example, the ASCII code for the letter "e" is represented as 101. Characters in a language like English appear with different probabilities. For instance, in an English book, there's an 8.2% chance that a randomly selected letter will be an "a" and only a 0.8% chance to find "k". Shannon's entropy, a concept from information theory, provides a way to quantify this information in bits. Represented as:

$$H_L = -\sum_{i=1}^{N_L} p_i \ log_2 (p_i) \qquad Eq. 9$$

Here, H_L is the Shannon entropy or information entropy. A language L consists of a finite set of N_L syllables, and p_i is the probability of the i-th syllable occurring (frequency). Shannon's entropy reaches its maximum when all syllables in language L are evenly distributed, meaning all p_i are equal. If p_i equals 1, the uncertainty disappears and H_L is 0. Shannon proved that one can never encode less than H_L bits of data on average.

For Lithuanian texts, we consider syllables as the basic language units. We calculate the probabilities of different syllables from a corpus of 80 Lithuanian texts to determine Shannon entropy rate. The Shannon entropy reaches a maximum when syllables are evenly distributed and drops to zero when there's no uncertainty.

In this study, information-theoretic approaches and language complexity estimations consider predictability distribution based on a relatively large number of linguistic data. Language is a communication channel between a speaker and a listener, and in this case, the difficulty of different Lithuanian texts can be defined by how much information this channel contains in bits with respect to syllables. By categorizing data into genres and individual texts I explore how information entropy measures vary, providing insights into the complexity and structure of Lithuanian language. This framework allows for the analysis of intuitive notions of language difficulty.

Conditional entropy

Shannon entropy offers a way to measure the average information content of syllables, treating each syllable independently of its context. This measure, while insightful, does not account for the dependencies that may exist between syllables in a language. In contrast, conditional entropy addresses this aspect by considering the context. It reveals the average amount of information needed to encode a syllable, given the knowledge of the syllable that precedes it. This context-aware measure is particularly relevant for understanding the relationships and dependencies between syllables, shedding light on the deeper structure of language. The formula for conditional entropy is given by:

$$H(X \mid C) = \sum_{c \in C} p(c) * H(X \mid C = c) =$$

$$= -\sum_{c \in C} p(c) * \sum_{i=1}^{N_L} p(X = x_i \mid C = c) \log_2(p(X = x_i \mid C = c)) \qquad Eq. 10$$

where the two random variables, C and X, represent the context and the syllable, respectively. The probability of a given context c within the range of values that C could choose is denoted by $p(c)$. The context is defined as the syllable that comes before. In this analysis the conditional entropy is denoted as $H(X_n \mid X_{n-1})$, meaning context is the preceding syllable.

For this analysis, I consider genres and individual texts as distinct contexts. This approach allows to compare information measures across different linguistic units and assess complexity within these subgroups. By doing so, it is possible to uncover the nuances of language that are not apparent when considering syllables in isolation. This method aligns with the methodology used by Pellegrino, Coupé, and Marsico (2007), who estimated phonological complexity using similar information-theoretic measures. Their work demonstrates the utility of these measures in capturing the intricate patterns of language use, and I apply part of those principles to this difficulty analysis of Lithuanian texts.

Classification of complex texts

To assess the complexity of literary texts, I have utilized entropy rates as a measure. 80 Lithuanian texts were categorized into 'Complex' and 'Not Complex' based on entropy rates, with the 75th percentile of entropy rates serving as the complexity threshold. This classification was executed in R, labelling texts as complex if their entropy rate surpassed the established threshold. To handle the diverse lengths of texts and standardize the input for modelling as well as the fact that we only have 80 Lithuanian texts, the texts were fragmented into equal-sized segments, each information about

containing 1000 syllables and those fragments with less than 1000 data points were removed. This method helps to avoid biases toward longer or shorter texts. Before creating fragments, data rows were shuffled in order to ensure that models are not biased to any inherited order. This shuffling was implemented by randomly reordering the data using the shuffle function.

Classification used these data features:

Unique Syllable Count: The number of unique syllables present in each fragment.

Average Syllable Frequency: The average frequency of occurrence of syllables within a fragment:

$$\frac{\sum \text{(Frequency of each unique syllable)}}{\text{Number of unique syllables}}$$

CV Pattern Uniqueness: The count of unique consonant-vowel (CV) patterns in each fragment.

Syllable Position Proportions: Proportions of syllables in positions initial, inner and end within each fragment.

Syllable Type Proportions: Proportions of different types of syllables: monosyllables (syllables make the whole word by themselves like “ir” “aš” and so on), disyllables (syllables that come from words that are made of two syllables like “da-rė”, “lė-kė” and so on) and polysyllables (syllables that come from words that have tree or more syllables like “pa-da-rė”, “lin-kė-ji-mai”) within each fragment.

Stop word Proportions: Proportions of stop word within each fragment.

CV Pattern Proportions: Proportions of consonant-vowel (CV) patterns in each fragment. This involves counting how many times a specific pattern (like CV, CVC, VC and so on) occurs within that fragment and divided by 1000 (fragment size).

I employed several classification algorithms for complex text classification: Gradient Boosting, Logistic Regression, Random Forest, Decision Tree, and K-nearest neighbours.

Results

Syllabic Bigram Network

In analysing the syntactic structure of the Lithuanian language, I used a first-order Markov chain model *(Eq. 5, 5.1)* to assess the occurrence and variability of syllables. To enhance the visual representation, a threshold was set to include only bigram combinations that appeared more than 2500 times, allowing to focus on the most frequent and structurally significant syllable pairs.

The resulting bigram network graph, depicted in Figure 1, reveals several key features of syllable distribution and connectivity. Central nodes, such as the syllable “si”, emerge prominently within the network, suggesting they may serve as common prefixes or suffixes in Lithuanian. These nodes often have a high degree of connectivity, indicating a tendency for syllable combination.

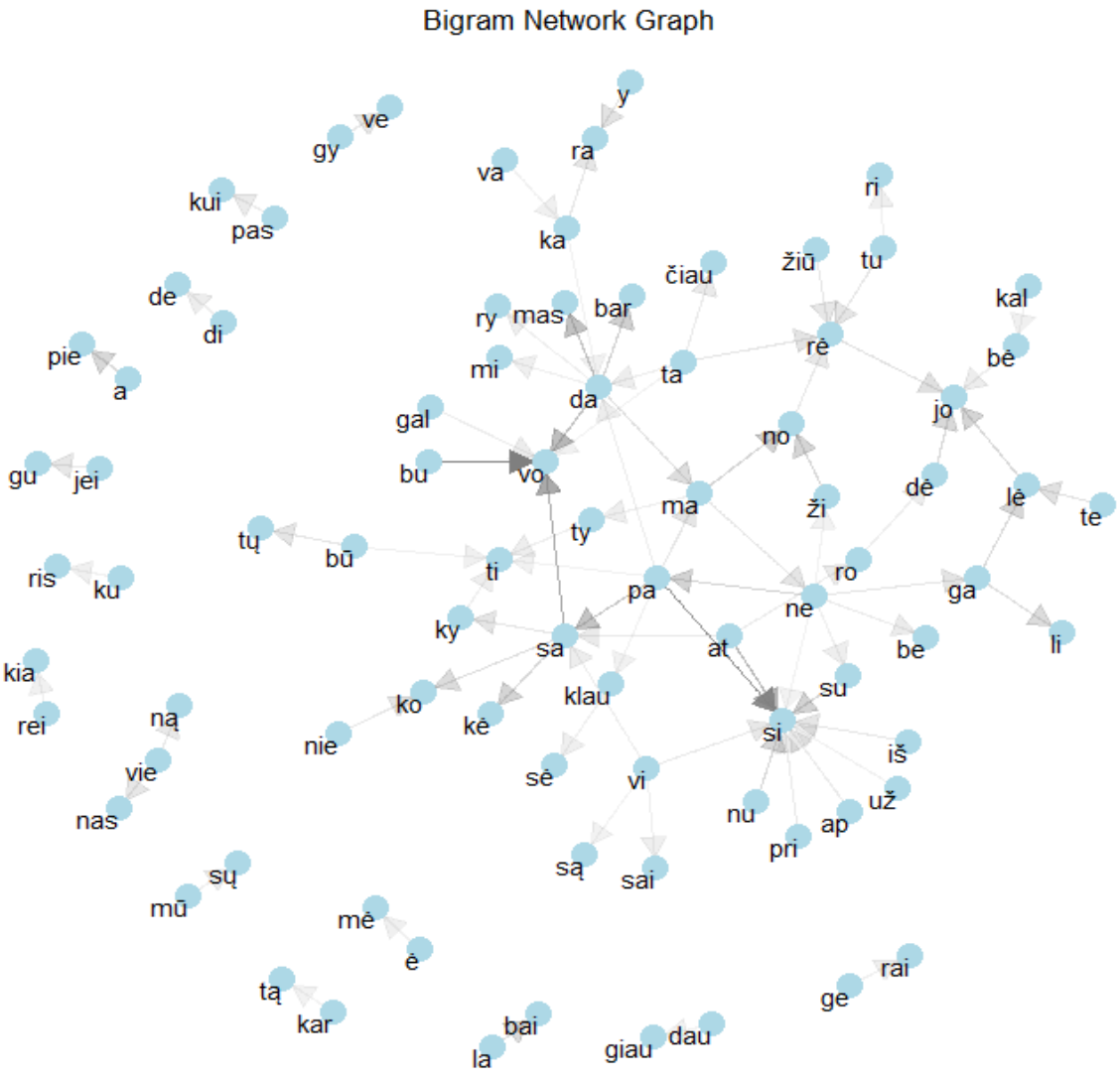


Figure 1. Syllable’s bigram network (frequency > 2500)

Thicker edges denote more frequent pairings, exemplified by the connection between “pa” and “si”, as opposed to thinner edges which represent fewer combinations like “pa” and “ti”. Additionally, certain syllables appear less connected or relatively isolated on the graph, hinting for their limited use or specificity in linguistic contexts. These less frequent nodes are indicative of syllables that may appear from commonly used words and are not prefixes or suffixes. Due to the frequent links between pairs of two-syllable nodes, the overall network architecture indicates that disyllabic words are common in Lithuanian.

Lithuanian language rank-frequency distribution of syllable

First to make any sense of the frequency of syllables and its rank, both sides have to be logarithmically transformed. Taking the logarithm of both sides gives us a chance to have a glimpse and inspect if a Zipf’s power-law distribution is nowhere visible, implying that it follows a relatively straight line on a doubly logarithmic plot. Please see Figure 2.

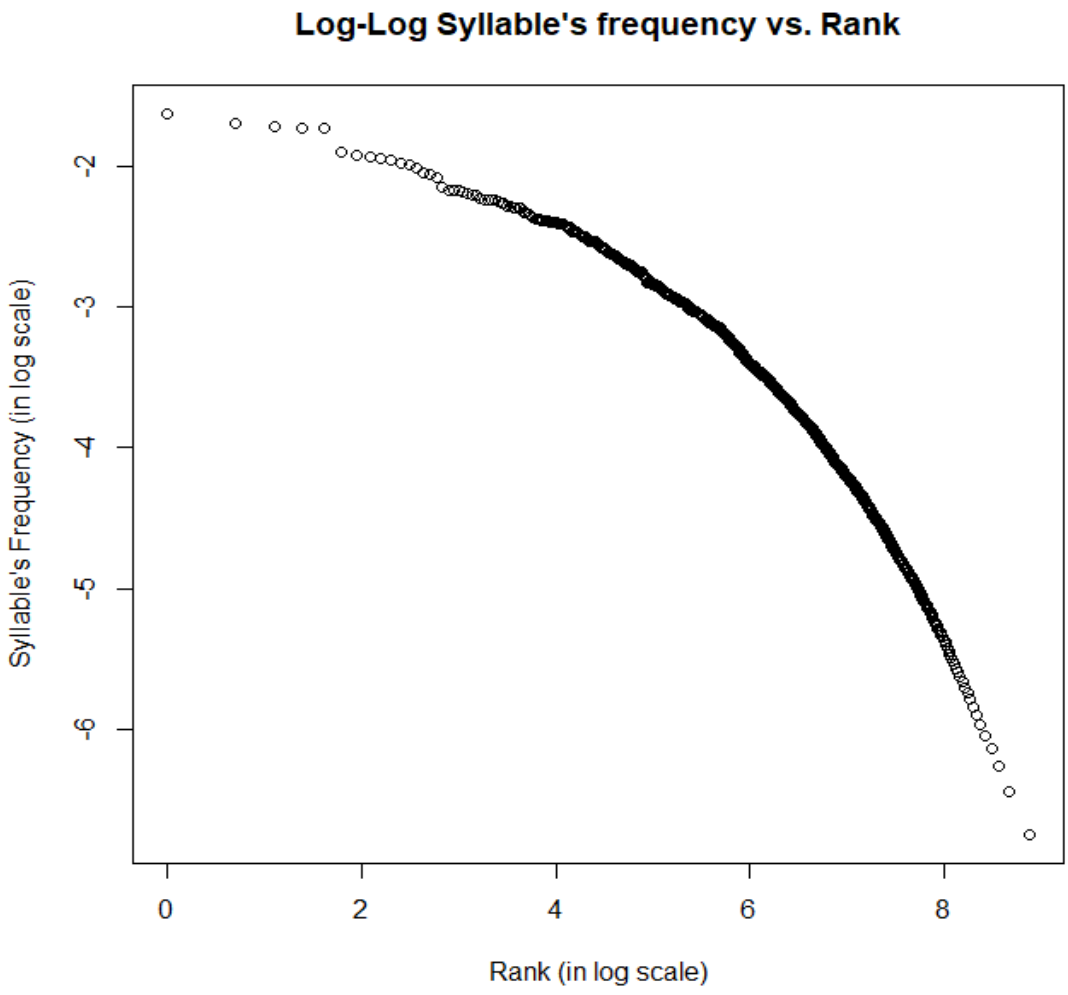


Figure 2. Logarithmically transformed Lithuanian syllable’s frequencies and rank.

From the Figure 2 it is visible that the slope becomes steeper towards the tail (high rank, least frequent syllables). Applied Zipf's power-law model (*Eq. 2*) is shown in the below Figure 3:

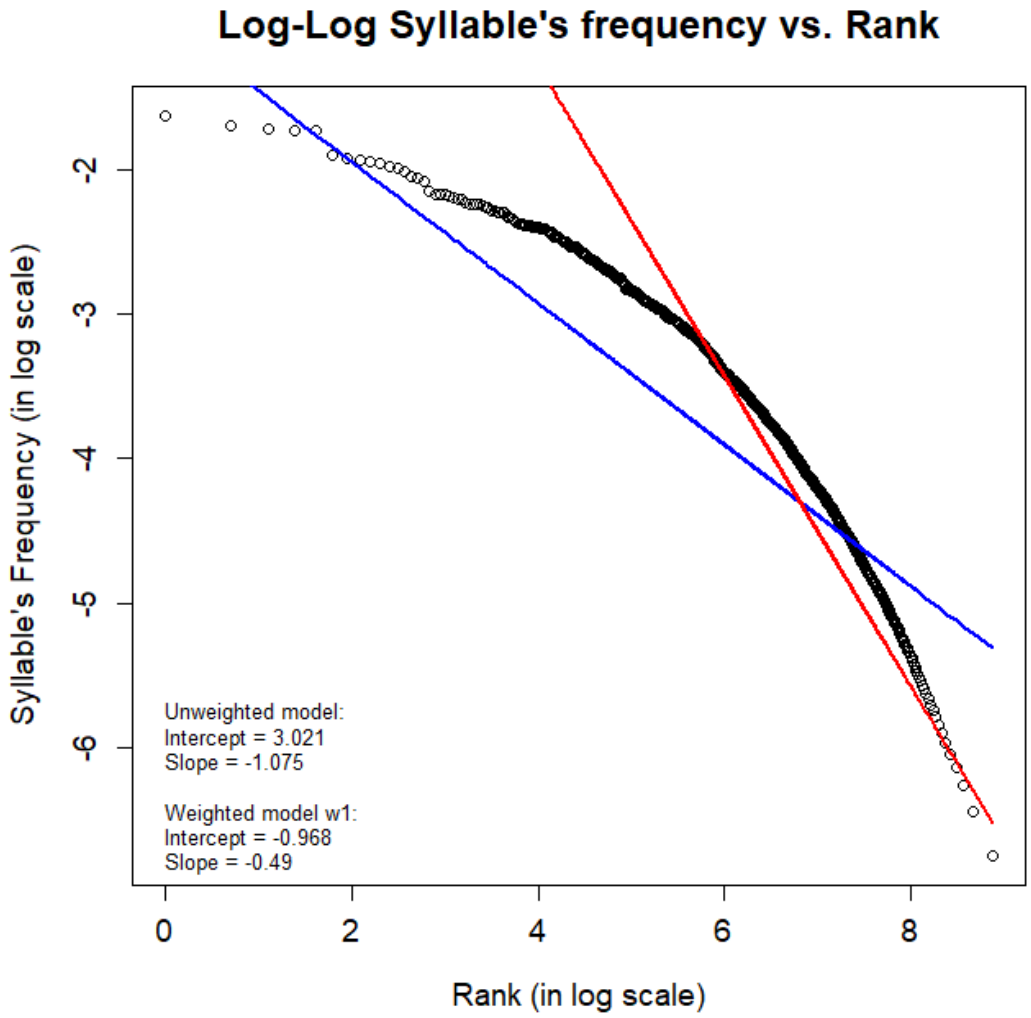


Figure 3. Logarithmically transformed Lithuanian syllable's frequencies and rank. Red line represents unweighted Zipf's laws regression fitting and blue weighted by scheme w_1.

Figure 3 clearly shows the observation that Zipf's law is not a good fit of the ranked syllables frequency distribution. This observation underlines the need for more complex models to accurately represent the distribution of syllable frequencies. A two-parameter fitting functions: Beta, Yule and Zipf-Mandelbrot *(Eq. 3, 4, 1* respectively*)* will be employed for that. The Beta model is characterized by the parameters $C, a, and\ b$, where C is a normalization constant, a affects the scaling for frequent syllables and b affects the scaling for less frequent ones. The estimated parameters for the unweighted Beta model are $C = 5.82$, $a = 0.96$, and $b = 0.06$. Similarly, the Yule model's parameters include C and a, which control the distribution's curvature, and b. For the unweighted Yule model, the estimated parameters are $C = 1.31, a = 0.62,$ and $b = 0.99$, indicating a different distribution shape compared to the Beta model. The Zipf model, with its simpler one-parameter form, is

represented by the scaling constant C and the exponent a. The estimated unweighted Zipf model parameters are $C = 20.51$ and $a = 1.07$, which suggest a steeper decline in frequency with rank than the two-parameter models. Lastly, the Zipf-Mandelbrot model, which is a generalization of the Zipf model, adds an additional parameter b to account for a shift in rank. The estimated parameters for this model are considerably higher, with $C = 3.25 * 10^8$, $a = 3.92$, and $b = 626.11$.

Table 1. presents a comparison of the Beta, Yule, Zipf and Zipf-Mandelbrot models, both in unweighted and weighted ($w_1 = \frac{1}{r}$) scenarios. The table outlines key statistical measures such as $SSE, AIC, \Delta AIC, and\ R^2$, providing a view of each model's performance.

Model	Unweighted					Weighted ($w_1 = \frac{1}{r}$)				
	K	SSE	AIC	ΔAIC	R^2	K	SSE	AIC	ΔAIC	R^2
Beta	3	398.34	-1415.43	14118.21	0.96	3	1,79	-68244,5	13601.95	0.91
Yule	3	69.84	-15533.65	0	0.99	3	0,34	-81846,5	0	0.98
Zipf	2	612.99	2077.80	17611.45	0.94	2	2,76	-64740,2	17106.29	0.86
Zipf-Mandelbrot	3	97.91	-12794,46	2739.188	-	3	1,23	-71307,6	10538.87	-

Table 1. Comparison of models in fitting ranked syllables frequency distribution obtained from Lithuanian texts.

The Yule model demonstrates superior performance across both weighted and unweighted models, with the highest R^2 values and the lowest SSE and AIC scores, indicating its effectiveness in capturing the rank-frequency distribution of Lithuanian syllables. Weighted regressions generally show improved fits, as it is indicated by reduced SSE values. This improvement highlights the significance of considering syllable rank in the model fitting process.

While the Yule model emerges as the best fit, the Beta and Zipf-Mandelbrot models also show decent performance, suggesting their potential utility in certain linguistic analysis contexts. Figures 3, 4, 6, and 7 visually represent the model fits and clearly illustrate the varying degrees of alignment between the models and the actual data. Figure 4 showcases the effectiveness of the Yule model, both in its unweighted and weighted forms.

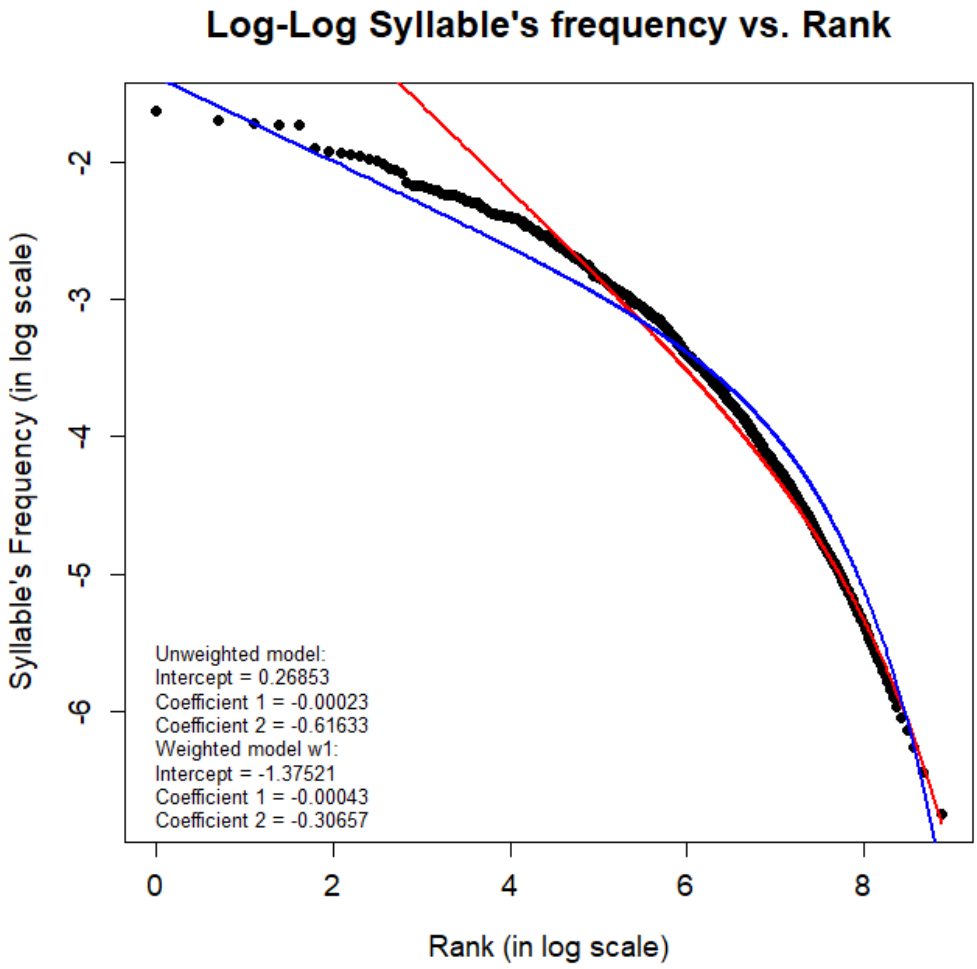


Figure 4. Logarithmically transformed Lithuanian normalized syllable's frequencies and rank. Red line represents unweighted Yule model regression fitting and blue weighted by scheme w_1.

Results from Table 1 and provided figures suggest that two-parameter models, particularly the Yule model, offer a more accurate representation of the syllable rank-frequency distribution in Lithuanian texts compared to the traditional one-parameter Zipf's law. This is significant for understanding the linguistic structure of the Lithuanian language and can inform further studies in linguistic complexity and syllable usage patterns.

Information entropy of Lithuanian language syllables

I have considered syllables as the symbols used by the system and estimated syllable occurring probabilities by counting the syllables and dividing it by the sample size. Shannon entropy quantifies the average information from the unigram model without considering any context. This allows to find how many bits take on average to encode linguistic variable. This formalized measure of information reduces the message to binary arithmetic coding in other words to 0 and 1. Shannon entropy estimations yielded an average value of 8.91 bits of information per syllable for the whole corpora of

80 Lithuanian texts syllables. Compared to Pellegrino et al. (2007) several languages study (Picture 1), the Lithuanian language would appear next to English with one of the highest Shannon entropy rates of 8.91.

Shannon entropy measure can be considered as well as the variability measure [6]. Shannon entropy measure study revealed how variability differs between 8 genres of 80 Lithuanian texts in terms of average information rate per syllable:

Genre	**Entropy rate**
Short story	8,84
Essay	8,87
Travel	**9,06**
Mythology	8,84
Tales	8,71
Plays	8,87
Poetry	**9,25**
Novel	8,78

Table 2. Syllable's Shannon entropy rate (in bits) for each genre

Calculations yielded an average value of 9.25 bits of information per syllable in the poetry genre, compared to 8,78 bits for the genre of the novel or 8,71 bits for the tales' genre.

Figure 4 presents a scatter plot that illustrates the relationship between the word count of various Lithuanian texts and their corresponding Shannon entropy values, segmented by genre. Shannon entropy, a measure of unpredictability or information content, indicates how evenly the syllables are distributed within a language. In the context of linguistic texts, a higher entropy value suggests a more uniform distribution of syllables, implying a richer and more complex usage of language like we have in poetry and travel genre compared to tales or short stories. Conversely, lower entropy may indicate a more predictable and less diverse syllable usage. However, it is important to note Shannon entropy rate increases not only because the syllable frequencies become more uniformed, but also because the number of different syllables (for example in longer texts) is higher. Thus, it is possible that long less rich text may have a higher entropy than a rich but short one.

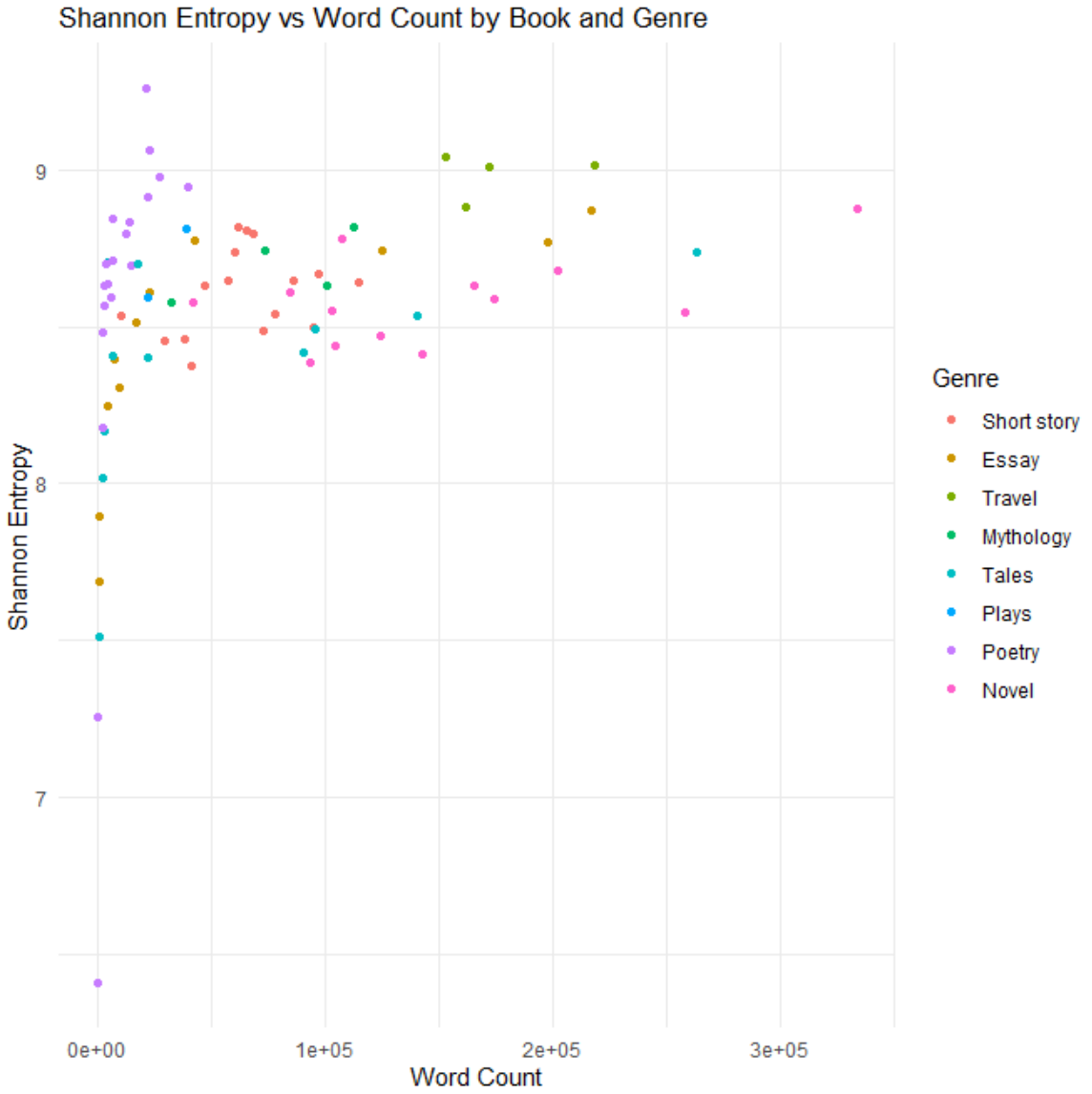


Figure 5. Shannon entropy rate vs. word count of each text grouped by genre.

From the plot, it appears that as the word count increases, the Shannon entropy values begin to cluster more closely. This suggests that larger text corpora provide a more stable estimation of entropy because they are less susceptible to the variability inherent in smaller samples. In other words, the larger the text, the more reliable the entropy measure becomes.

In Figure 5, it is evident that entropy values vary across different genres, with poetry characteristically exhibiting higher entropy rates and a notable degree of variability. Such a trend correlates with the inherent stylistic nuances of poetry, which typically uses a broad vocabulary and innovative language usage. In contrast, genres such as tales and short stories are often associated with lower entropy values, potentially reflecting their narrative straightforwardness and a tendency towards repetition.

It is worth to mention that texts with a higher count of unique syllables generally display increased Shannon entropy. This pattern is underpinned by a moderate positive correlation between entropy and

word count in the texts. Longer compositions encompass a wider syllable variety, which, in turn, increases entropy rate. This phenomenon suggests that even a text with a comparatively sparse vocabulary could manifest a high entropy if it is lengthy, thereby countering the presumption that only lexically rich texts exhibit high entropy. Spearman's rank correlation coefficient was approximated to 0.398 (p-value = 0.0002855), this lends statistical support to this observation. The coefficient indicates that as texts expand in length, there is a tendency towards a more varied syllable usage, contributing to the increase in entropy. However, the moderate strength of this correlation suggests the presence of additional factors influencing entropy.

The table 4 provides lists of specific texts along with their genres and entropy rates. At the top of the entropy values are poetry works such as "Judita_VAICIUNAITE__Eilerasciai" with an entropy rate of 9.26, indicating a highly diverse syllable distribution within the text. Travel literature such as "Ligi_Lietuvos_10000_kilometru" and "Kelionė_į_Sambalą" also show high entropy rates, suggesting a richness in language that might be attributed to the descriptive travel writing, because they are full of location names and foreign words with atypical syllables. At the other end of the spectrum, texts like "Brisiaus_galas," "Kliudžiau," and "Juratė_ir_Kastytis" have lower entropy values, which may reflect a narrower syllable distribution or a more repetitive use of language, common in stories aimed at children or in simpler narratives.

Conditional entropy

Shannon entropy provides an estimate of the average uncertainty in predicting a syllable from a distribution that is observed within a large body of text. It is a measure of the unpredictability inherent in the syllable structure of a language without any given context. Conversely, conditional entropy *(Eq. 10)* measures the residual uncertainty of a syllable's occurrence when the preceding syllable is known. Given that words are typically composed of multiple syllables, the presence of context reduces uncertainty. Naturally, conditional entropy tends to be lower than Shannon entropy, reflecting the reduction in unpredictability provided by this contextual information.

Genre	**Conditional entropy rate)**
Short story	6,26
Essay	6,25
Travel	**6,32**
Mythology	6,11
Tales	6,06
Plays	5,79
Poetry	**6,55**
Novel	6,20

Table 3. Syllable’s conditional entropy rate (in bits) for each genre. Conditional here is the preceding syllable.

Conditional entropy may reveal how certain syllables are predisposed to appear at the beginning (like prefixes) or at the end (like suffixes) of words. This distribution is not uniform, preceding syllables often carry more information, making them "information-rich" [17]. In contrast, Shannon entropy measures the overall diversity and frequency of syllable usage throughout a text corpus. In a hypothetical, simplified scenario where every syllable is equally probable what one might consider ideal state each syllable would be used inversely proportional to the effort required to articulate it. This would represent a language of maximum efficiency and minimal predictability, a model that serves as a theoretical baseline rather than a reflection of actual linguistic complexity.

In the context of this analysis, the total conditional entropy has been calculated to be approximately 6.45. This value quantifies the average unpredictability of a syllable sequence within a given context of previous syllable and underscores the complexity of understanding syllable usage when previous or subsequent syllables are considered. Compared to other language conditional entropy findings based on syllable’s as linguistic units, Lithuanian language would be somewhere between German (6.08) and French (6.68) as it was estimated in [19]. For more details see Picture 3 in appendices.

Our findings indicate that the Shannon entropy of Lithuanian texts provides insight into the overall complexity of the language concerning its syllables. By employing both unigram models and context-aware conditional entropy measures, I observed variability in the information content across different genres and texts. This variability reflects the richness and diversity of the language's syllabic structure. Further, the results underscore the intricate balance between frequency and information content, where frequently occurring syllables carry less information, consistent with the principles of Zipf's law. This balance suggests an optimization of the language's communicative efficiency, where the most common messages are transmitted with the least amount of information, avoiding redundancy while maintaining clarity.

Complex text‘s classification

The complexity of literary texts was determined using entropy rates as a measure of complexity. A significant aspect of this analysis was the classification of texts into 'Complex' or 'Not Complex' categories, based on their entropy rates. The threshold for difficulty was established as the 75th percentile (4th quantile) of entropy rates among the texts. This quantile-based approach ensures that texts with high variability and unpredictability in their language usage are categorized as 'Complex'. The labelling process in R involved comparing each text's entropy rate against this threshold and assigning the corresponding complexity label. For the classification task, the gradient boosting, logistic regression, random forest, decision tree and k-nearest neighbours (KNN) classification algorithms were employed. In terms of accuracy gradient boosting classification model performed the best with

91% accuracy. Gradient Boosting is an ensemble learning technique that builds a series of decision trees sequentially. Each tree in the sequence focuses on correcting the mistakes of the previous one. The model operates on the principle:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Where $F_m(x)$ is the model at iteration m, γ_m is the learning rate, and $h_m(x)$ is the decision tree. The final model is a weighted sum of these weak learners, aiming to minimize the deviance loss function. Deviance loss for a binary classification is:

$$L(y, p) = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where y is the true label, p is the predicted probability, and n is the number of samples.

The models' performance was assessed using metrics such as precision, recall, f1-score, and the confusion matrix.

```
Model: Gradient Boosting
Classification Report:
              precision    recall  f1-score   support

     Complex       0.87      0.88      0.88       381
 Not Complex       0.94      0.93      0.93       723

    accuracy                           0.91      1104
   macro avg       0.90      0.91      0.91      1104
weighted avg       0.91      0.91      0.91      1104

Confusion Matrix:
[[337  44]
 [ 51 672]]
```

The Gradient Boosting model had a precision of 94% for 'Not Complex' texts and 87% for 'Complex' texts, indicating its high accuracy in predicting both categories. This model achieved a recall of 93% for 'Not Complex' and 88% for 'Complex', showing how effectively it can identify most of the true 'Complex' and 'Not Complex' cases. The f1-scores for both categories (93% for 'Not Complex' and 88% for 'Complex') reflect the model's balanced precision and recall. The Gradient Boosting model's confusion matrix showed 672 true positives (correctly identified not complex texts), 51 false positives (complex texts incorrectly identified as not complex), 44 false negatives (not complex texts incorrectly identified as complex), and 337 true negatives (correctly identified complex texts). Less accurate or balanced classification results of Logistic Regression (accuracy – 90%), Random Forest (accuracy – 91%), Decision Tree (88%) and KNN (87%) models can be reviewed in Table 6.

The results from the classification models not only provided a clear distinction between complex and not complex texts but most important highlights features that influence this classification as it offers insights into which aspects of a text most strongly relate to its perceived complexity. This understanding is beneficial for both the analysis and the development of more refined models in the future. From the table 5 it is visible that most influential feature is 'unique_syllable_count,' with an importance score of ~0.79. This strongly suggests that the diversity of syllables within a text plays a crucial role in determining its difficulty. Other notable features include various consonant-vowel CV patterns (like 'cv_pattern_CV' and 'cv_pattern_CCVC'), indicating that the complexity of syllabic structures also contributes to the overall complexity of the text. 'avg_syllable_freq' also emerged as a contributing feature, emphasizing the importance to text difficulty of how frequently certain syllables occur within a text.

Feature	**Importance**
unique_syllable_count	0.78976
cv_pattern_CV	0.04458
avg_syllable_freq	0.03961
type_monosyllables	0.02211
position_inner	0.01637
type_disyllables	0.01065
type_polysyllables	0.00975
stopai_N	0.00898
cv_pattern_CCVC	0.00855
cv_pattern_CVC	0.00660

Table 5. Feature importance ccores for text complexity classification

Discussion

Tambovtsev et al. (2007) whom also included the Lithuanian language in their study of phoneme's rank-frequency distribution and came to the conclusion that the Yule function fits the distribution of phoneme frequencies better than Zipf's law and several other models. Since Yule function fits our Lithuanian language text's syllable's rank-frequency data best compared to several other commonly used in quantitate linguistics rank-frequency models, this analysis raises a new discussion point if the syllable's rank-frequency distribution behaves more like phonemes than words. It would be interesting to include phonemes and more models into the analysis in order to address the assumption of syllable's rank-frequency behaviour similarity to phonemes.

Conclusions

In this thesis, I applied various distribution models to syllabified Lithuanian text data. Both the Beta and Yule models, as well as the Zipf-Mandelbrot model, better fit the Lithuanian language syllable's rank-frequency distribution compared to Zipf's law. The best fit was achieved using monotonically decreasing weights that assign higher weights to high-frequency syllables. Since Yule model achieved the best fitting, this raises a new discussion point: whether syllables behave more like phonemes than words in terms of their ranked occurrence probabilities, as a few studies have found that the Yule model describes phoneme frequencies best [5].

The bigram network analysis, conducted using Markov Chains, revealed an intertwined network of syllables within the Lithuanian language. This network mostly consists of syllables that are prefixes and suffixes, suggesting their frequent use in word formations. Additionally, the analysis identified a distinct set of syllables that appear disconnected from this intertwined network, indicating their unique use in the language corpus. This separation in the syllable network shows the diverse structural elements in Lithuanian language and their varying roles in linguistic construction.

In the second part of the thesis, I considered the Shannon entropy measure and conditional entropy as measures of difficulty and variability [6]. This part of the study revealed how variability differs among 8 genres of 80 Lithuanian texts in terms of the average information rate per syllable and how much average information in bits can be expected from different types of texts in terms of syllables. It takes an average of 9.25 bits of information to encode a syllable in the poetry genre, compared to 8.78 bits for novels and 8.71 bits for tales. The Shannon entropy measure yielded an average value of 8.91 bits of information per syllable for the entire corpus of 80 Lithuanian texts. Compared to Pellegrino et al. (2007)'s study of several languages (Picture 1), the Lithuanian language appears next to English, exhibiting one of the highest Shannon entropy rates. Conditional entropy, given the context of the preceding syllable, revealed an entropy rate of 6.45. Compared to other languages estimated in [19], Lithuanian would appear somewhere between German and French.

In an attempt to understand more about text complexity, I classified chunks of texts based on syllabic information into "Complex" and "Not Complex" categories. The gradient boosting classification algorithm was the most accurate, with a 91% accuracy rate. This part of the study revealed that most text fragments could be classified as more difficult (requiring more effort) based on the unique number of syllables they contain.

The analysis presented in this thesis offers contributions to the understanding of the Lithuanian language's syllabic difficulty and distribution. It provides some valuable insights for linguistics applications, particularly in language frequency distribution modelling and text difficulty analysis.

References

[1] Rujević, Biljana, et al. "Quantitative analysis of syllable properties in Croatian, Serbian, Russian, and Ukrainian." Language and Text: Data, models, information and applications 356 (2021): 55.

[2] Radojičić, Marija, et al. "Frequency and length of syllables in Serbian." Glottometrics (2019).

[3] Li, Wentian, Pedro Miramontes, and Germinal Cocho. "Fitting ranked linguistic data with two-parameter functions." Entropy 12.7 (2010): 1743-1764.

[4] Miestamo, Matti, Kaius Sinnemäki, and Fred Karlsson, eds. Language complexity: Typology, contact, change. Vol. 94. John Benjamins Publishing, 2008. (page 91)

[5] Tambovtsev, Yuri, and Colin Martindale. "Phoneme frequencies follow a Yule distribution." SKASE Journal of Theoretical Linguistics 4.2 (2007): 1-11.

[6] Carcassi, Gabriele, Christine A. Aidala, and Julian Barbour. "Variability as a better characterization of Shannon entropy." European Journal of Physics 42.4 (2021): 045102.

[7] Pellegrino, François, Christophe Coupe, and Egidio Marsico. "An information theory-based approach to the balance of complexity between phonetics, phonology and morphosyntax." 81st Annual Meeting of the Linguistic Society of America 2007. Anaheim, CA, 2007.

[8] Martindale, Colin, et al. "Comparison of equations describing the ranked frequency distributions of graphemes and phonemes." Journal of Quantitative Linguistics 3.2 (1996): 106-112.

[9] Strauss, Udo, F. Fan, and G. Altmann. "Problems in Quantitative Linguistics 1." Glottotheory 2.1 (2009): 141-142.

[10] Wimmer, Gejza, and Gabriel Altmann. “Thesaurus of univariate discrete probability distributions. Stamm”, 1999.

[11] Zipf, G. K. "Human Behavior attd the Principle of Least Effort.", 1949.

[12] Tanaka-Ishii, Kumiko. Statistical Universals of Language: Mathematical Chance vs. Human Choice. Springer Nature, 2021. (Chapter 10, Page 102)

[13] Kazlauskienė, Asta, Gailius Raškinis, and Airenas Vaičiūnas. "Automatinis lietuvių kalbos žodžių skiemenavimas, kirčiavimas, transkribavimas." (2010).

[14] Shannon, C. E. (1948). A mathematical theory of communication. Bell System Technical Journal, 27, 379-423, 623-656.

[15] Schwartz M. Statistical Mechanics, Lecture 6: Entropy, Spring 2019. available at: https://scholar.harvard.edu/files/schwartz/files/6-entropy.pdf

[16] Kasparaitis P. Kompiuterinė lingvistika. Skiemenavimas ir žodžių kėlimas (angl. hyphenation). 2005

[17] Murauskas, G., & Radavičius, M. (2022). Skiemenų statistikos taikymas atskiriant poeziją nuo prozos. Lietuvos statistikos darbai, 61, 32-45.

[18] Mandelbrot, B. (1953). An informational theory of the statistical structure of language. Communication theory, 84, 486-502.

[19] Oh, Y. M. (2015). Linguistic complexity and information: Quantitative approaches. Unpublished doctoral thesis, University of Lyon.

APPENDICES

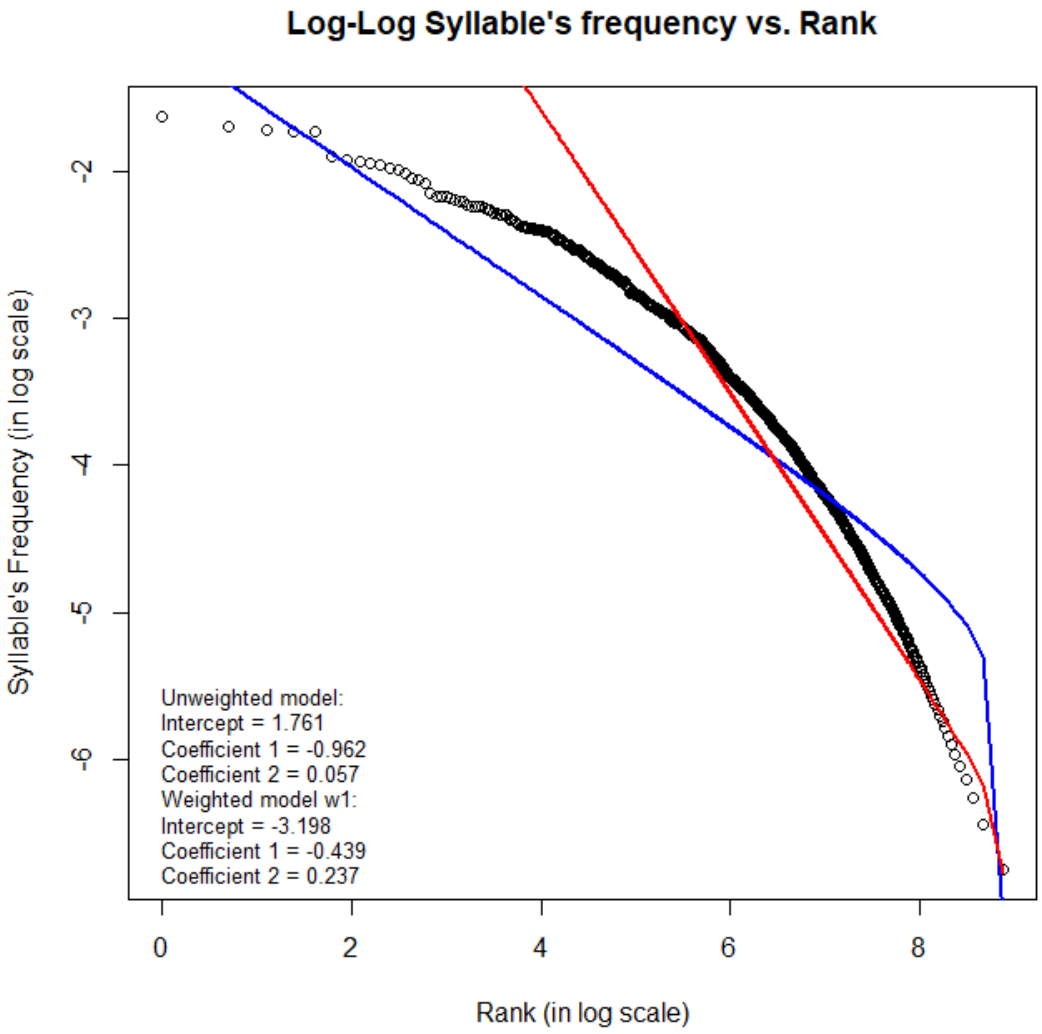


Figure 6. Logarithmically transformed Lithuanian syllable's frequencies and rank. Red line represents unweighted Beta model regression fitting, blue weighted by scheme w_1.

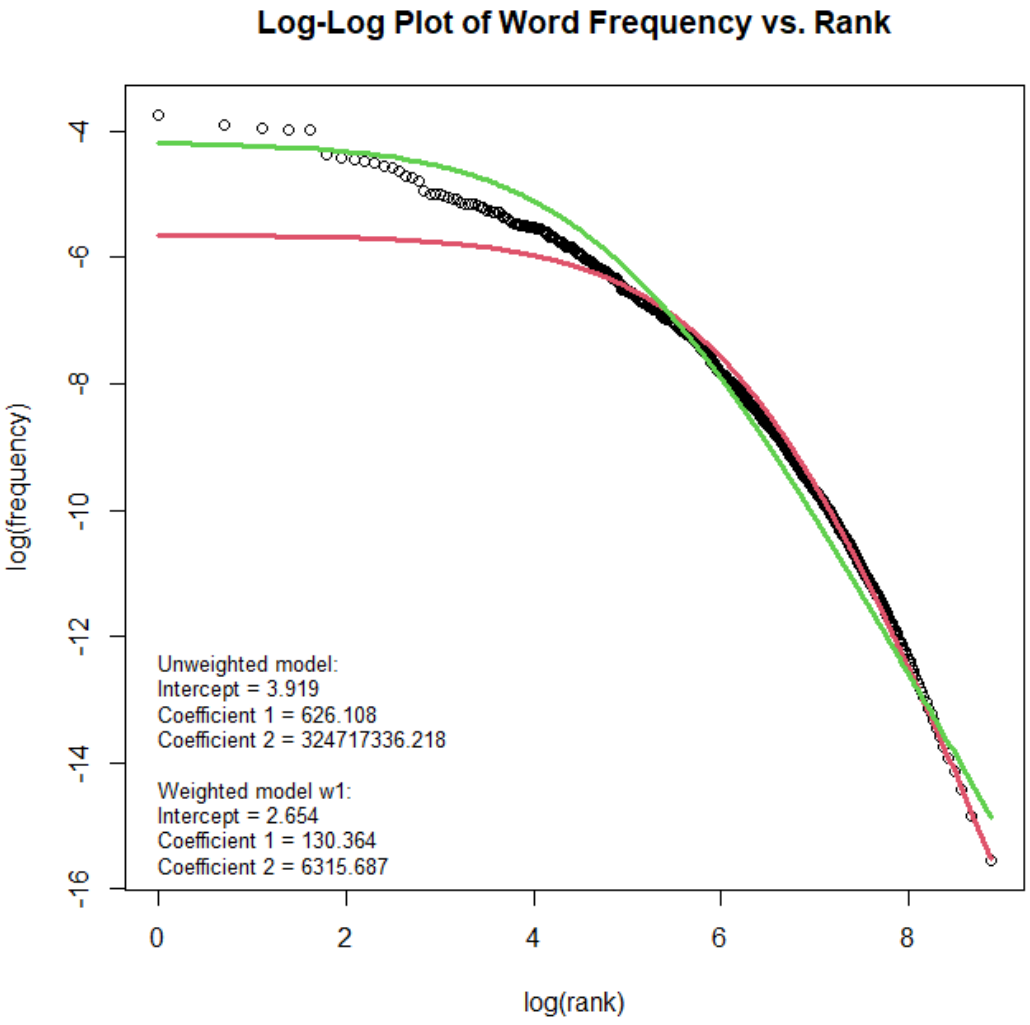


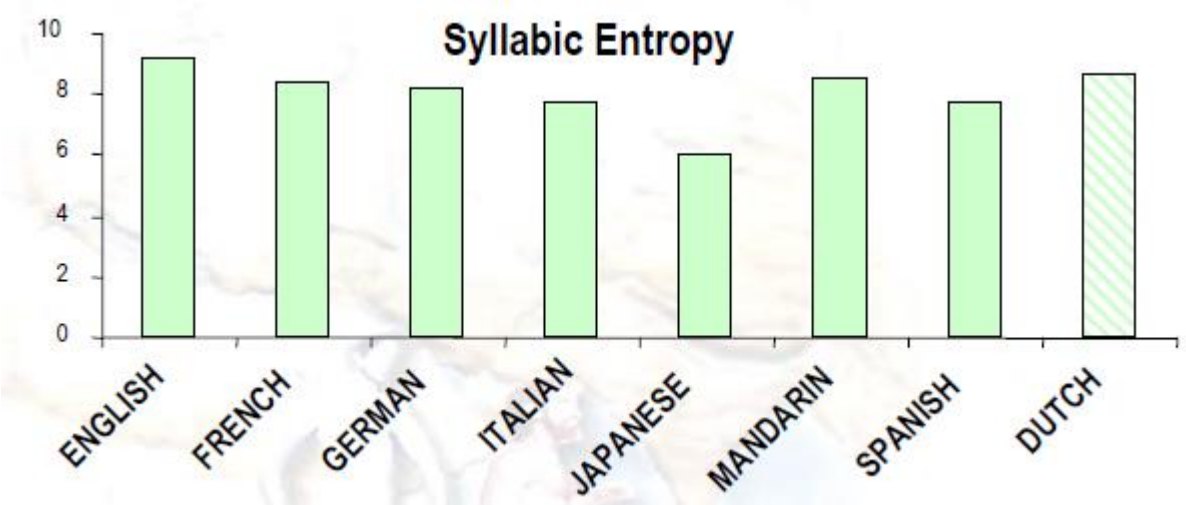
Figure 7. Logarithmically transformed Lithuanian syllable's frequencies and rank. Red line represents unweighted Zipf-Mandelbrot model fitting, green weighted by scheme w_1.

Text/Book	**Genre**	**Entropy rate**
Judita_VAICIUNAITE__Eilerasciai	Poetry	9,26
Konradas_Valenrodas	Poetry	9,06
Ligi_Lietuvos_10000_kilometru	Travel	9,04
Kelione_i_Sambala	Travel	9,02
Kelioniu_alchemija	Travel	9,01
Pavasario_balsai	Poetry	8,98
Metai	Poetry	8,94
Bobute_is_Paryziaus_arba_Lakstingala_Zarasuose	Poetry	8,91
Keliones	Travel	8,88
Raudonosios_mirties_kauke	Novel	8,88
Link_Debesijos	Short story	8,87
Pauksciu_takas	Poetry	8,84
Bilietas_is_dangaus_arba_Jono_Grigo_kelione_greituoju_traukiniu	Poetry	8,83

Naktys_Karaliskiuose	Essay	8,82
Graiku_mitai	Mythology	8,82
Romeo_ir_Dzuljeta	Play	8,81
Kintas	Essay	8,81
Grazina	Poetry	8,80
Zuika_padukelis	Essay	8,80
Gugis_-_giriu_kaukas_ir_zmoniu_draugas	Novel	8,78
Nebijoke	Short story	8,77
Papartynu_saule	Short story	8,77
Padavimai_ir_legendos	Mythology	8,74
Uzrasai_apie_Serloka_Holmsa	Short story	8,74
Haufo_pasakos	Tale	8,74
Burimas_obuolio_seklom	Essay	8,74
Giesme_apie_Gedimina	Poetry	8,71
Egle_Zalciu_karaliene	Tale	8,71
Vilniaus_legendos_Pasakos	Tale	8,70
Giedancios_upes	Poetry	8,70
Baltoji_varnele	Poetry	8,70
Dedes_Tomo_trobele	Novel	8,68
Dainavos_salies_senu_zmoniu_padavimai	Essay	8,67
BASKERVILIU_SUO	Essay	8,65
Zydrieji_Jungos	Essay	8,65
Tomo_Sojerio_nuotykiai	Essay	8,64
Nebaigta_Tale	Poetry	8,64
Perkuno_sventykloje	Essay	8,63
Ane_is_Zaliastogiu	Novel	8,63
Rugseji_sermuksnio_uoga	Poetry	8,63
Suzeistas_vejas	Mythology	8,63
Vasara_su_Katsuniu	Novel	8,61
Jura_yra_suri	Short story	8,61
Pelenu_antele	Play	8,60
Anksti_ryta	Poetry	8,59
Robinzonas_Kruzas	Novel	8,59
kurio_nieks_nemylejo	Novel	8,58
Kaip_atsirado_zeme	Mythology	8,58
Zmogaus_zvaigzde	Poetry	8,57
Portugalijos_karalius	Novel	8,55
Algimantas	Novel	8,54
Ka_nuveike_Keite	Essay	8,54
Vaiku_karas	Essay	8,54
Lietuviskos_pasakos	Tale	8,53
Skerdzius	Short story	8,52
Bembio_vaikai	Essay	8,50
Broliai_Grimai_Vaiku_ir_namu_pasakos	Tale	8,49
Jurgio_Paketurio_klajones	Essay	8,49
Vejo_birbyne	Poetry	8,48
Paslaptingas_sodas	Novel	8,47
Alisa_stebuklu_salyje	Essay	8,46

Jos_vardas_Nippe	Essay	8,46
Baltaragio_malunas_arba_Kas_dejosi_anuo_metu_Paudruves_kraste	Novel	8,44
Po_raganos_kirviu_Senas_kareivis_Matatutis	Tale	8,42
Heida	Novel	8,41
Nidos_zverys	Tale	8,41
Mazasis_princas	Tale	8,40
Galvazudys	Short story	8,40
Poliana	Novel	8,39
Bembis	Essay	8,37
Irkos_tragedija	Short story	8,30
Kudikystes_sapnai	Short story	8,25
Sigitas_PARULSKIS__Eilerasciai	Poetry	8,18
Auksine_kurpele	Tale	8,17
Laimes_ziburys	Tale	8,02
Brisiaus_galas	Short story	7,90
Kliudziau	Short story	7,69
Jurate_ir_Kastytis	Tale	7,51
Eilerasciai	Poetry	7,25
Peizazas	Poetry	6,41

Table 4. Syllable's Shannon entropy rate (in bits) for each text/book



Picture 1. Source: Pellegrino, François, Christophe Coupe, and Egidio Marsico. "An information theory-based approach to the balance of complexity between phonetics, phonology and morphosyntax." 81st Annual Meeting of the Linguistic Society of America 2007. Anaheim, CA, 2007.

e	t	a	o	i	n	s	h	r	d	l	u	c
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8
m	w	f	y	g	p	b	v	k	x	j	q	z
2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.2	0.2	0.1	0.1

Picture 2. Source [15]: https://scholar.harvard.edu/files/schwartz/files/6-entropy.pdf

Table 2.11: $H(X_n|X_{n-1})$ and $H(X_n|X_{n+1})$. The maximum and minimum values are marked in green and blue.

Language	$H(X_n\|X_{n-1})$	$H(X_n\|X_{n+1})$
CAT	5.49	5.53
CMN	6.96	6.99
DEU	6.08	6.13
ENG	7.09	7.10
EUS	4.83	5.05
FIN	5.49	5.86
FRA	6.68	6.76
HUN	5.90	5.95
ITA	5.29	5.26
JPN	5.03	5.07
KOR	5.56	5.53
SPA	5.43	5.41
SRP	5.47	5.99
THA	7.19	7.13
TUR	5.34	5.18
VIE	8.02	8.04
YUE	6.53	6.59

Picture 3. Source: Oh, Y. M. (2015). Linguistic complexity and information: Quantitative approaches. Unpublished doctoral thesis, University of Lyon. (Page 61)

```
Model: Logistic Regression
Classification Report:
              precision    recall  f1-score   support

     Complex       0.87      0.82      0.84       381
 Not Complex       0.91      0.93      0.92       723

    accuracy                           0.90      1104
   macro avg       0.89      0.88      0.88      1104
weighted avg       0.90      0.90      0.90      1104

Confusion Matrix:
[[313  68]
 [ 47 676]]
```

```
Model: Random Forest
Classification Report:
              precision    recall  f1-score   support

     Complex       0.89      0.86      0.87       381
 Not Complex       0.93      0.94      0.94       723

    accuracy                           0.91      1104
   macro avg       0.91      0.90      0.91      1104
weighted avg       0.91      0.91      0.91      1104
```

```
Confusion Matrix:
[[328  53]
 [ 41 682]]
```

```
Model: Decision Tree
Classification Report:
              precision    recall  f1-score   support

     Complex       0.82      0.82      0.82       381
 Not Complex       0.90      0.91      0.91       723

    accuracy                           0.88      1104
   macro avg       0.86      0.86      0.86      1104
weighted avg       0.88      0.88      0.88      1104

Confusion Matrix:
[[312  69]
 [ 67 656]]
```

```
Model: KNN
Classification Report:
              precision    recall  f1-score   support

     Complex       0.81      0.82      0.82       381
 Not Complex       0.90      0.90      0.90       723

    accuracy                           0.87      1104
   macro avg       0.86      0.86      0.86      1104
weighted avg       0.87      0.87      0.87      1104

Confusion Matrix:
[[312  69]
 [ 71 652]]
```

Table 6. Other classification model metrics for complex and not complex text fragments.

CODE:

```
library(dplyr)
library(tidyr)
library(tidyverse)
library(scales)
library(ggplot2)
library(tidytext)
library(igraph)
library(widyr)
library(igraph)
library(ggraph)
library(gridExtra)
library(grid)
library(purrr)
set.seed(123)


#### Data preprocessing and cleaning

long_syldata <- syldata %>%
  gather(Syllablesno, Syllables, SK1:SK12)


long_syldata <- long_syldata %>%
  filter(Syllables != "<NA>")
### CV coding
is_vowel <- function(letter) {
  return(tolower(letter) %in% c('a', 'ą', 'e', 'ė', 'ę', 'i', 'į','y', 'o', 'u', 'ū', 'ų'))
}
convert_to_cv <- function(syllable) {
  letters <- strsplit(syllable, "")[[1]]
  cv_code <- sapply(letters, function(letter) if(is_vowel(letter)) "V" else "C")
  return(paste(cv_code, collapse = ""))
}


long_syldata <- long_syldata %>%
  mutate(cv_pattern = sapply(Syllables, convert_to_cv))


cv_grouped <- long_syldata %>%
  group_by(cv_pattern) %>%
  summarise(
    count = n(),
    examples = list(unique(Syllables)),
    .groups = 'drop'
  )


### removing incorrect syllables patterns

#pattern is wrong if it has V separated by one or more C or if it consists only of C like VCVC
```

```
is_invalid_syllable <- function(cv_pattern) {
  invalid_pattern <- str_detect(cv_pattern, "VC+V") || !str_detect(cv_pattern, "V") ||
str_detect(cv_pattern, "^C+$")
  return(invalid_pattern)
}

invalid_mskiem_groups <- long_sylldata %>%
  mutate(is_invalid = sapply(cv_pattern, is_invalid_syllable)) %>%
  filter(is_invalid) %>%
  distinct(MSkiem)


cleaned_sylldata <- long_sylldata %>%
  filter(!(MSkiem %in% invalid_mskiem_groups$MSkiem))


cv_grouped_cleaned <- cleaned_sylldata %>%
  group_by(cv_pattern) %>%
  summarise(
    count = n(),
    examples = list(unique(Syllables)),
    .groups = 'drop'
  )


cleaned_sylldata <- cleaned_sylldata %>%
  add_count(Syllablesno, Syllables, sort = TRUE) %>%
  mutate(SKno_ranks = min_rank(desc(n)))


cleaned_sylldata <- cleaned_sylldata %>%
  add_count(Syllables, sort = TRUE) %>%
  mutate(totalcount_ranks = min_rank(desc(nn)))


cleaned_sylldata <- cleaned_sylldata %>%
  dplyr::rename(count_SKno = n, totalcount = nn) %>%
  mutate(Syll_length = nchar(Syllables)) %>%
  filter(Syll_length < 10)


#### number of words

unique_words_count <- cleaned_sylldata %>%
  distinct(word) %>%
  nrow()


##### number of unique syllables

unique_syllables_count <- cleaned_sylldata %>%
  distinct(Syllables) %>%
  nrow()


### Sylables structure (CV pattern) distribution
```

```
genres <- unique(cleaned_syldata$zanras)


genre_plots <- list()

for (genre in genres) {
  genre_data <- cleaned_syldata %>%
    filter(zanras == genre) %>%
    count(cv_pattern, name = "frequency") %>%
    arrange(desc(frequency))


  genre_plots[[genre]] <- ggplot(genre_data, aes(x = reorder(cv_pattern, -frequency), y = frequency)) +
    geom_bar(stat = "identity") +
    labs(title = paste("Frequency of Syllable Types in", genre),
         x = "Syllable Type",
         y = "Frequency") +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
}
grid_plots <- do.call(grid.arrange, c(genre_plots, ncol = 2, nrow = 4))


#### creating syllables types

cleaned_syldata <- cleaned_syldata %>%
  mutate(Syll_type = case_when(
    Tsyll == 1 ~ 'monosyllables',
    Tsyll == 2 ~ 'disyllables',
    Tsyll >= 3 ~ 'polysyllables'
  ))


cleaned_syldata <- cleaned_syldata %>%
  mutate(Syll_position = as.numeric(gsub("SK", "", Syllablesno)))


# categorizing syllable position

cleaned_syldata <- cleaned_syldata %>%
  mutate(Syll_position_group = case_when(
    Tsyll == 1 ~ 'initial',
    Syll_position == 1 & Tsyll >= 2 ~ 'initial',
    Tsyll == Syll_position ~ 'end',
    TRUE ~ 'inner'
  ))


# n-grams analysis (bigram network)


make_bigrams <- function(...) {
  syllables <- na.omit(c(...))
  bigrams <- map2_chr(head(syllables, -1), tail(syllables, -1), ~ paste(.x, .y))
  return(bigrams)
}
```

```
bigram_counts <- syldata %>%
  transmute(syllables = pmap(list(SK1, SK2, SK3, SK4, SK5, SK6, SK7, SK8, SK9, SK10, SK11,
SK12), make_bigrams)) %>%
  unnest(cols = c(syllables)) %>%
  separate(syllables, into = c("syllable_sequence", "next_syllable"), sep = " ") %>%
  count(syllable_sequence, next_syllable, sort = TRUE)


bigram_graph <- bigram_counts %>%
  filter(n > 2500) %>%
  graph_from_data_frame()


a <- grid::arrow(type = "closed", length = unit(0.15, "inches"))



bigram_plot <- ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(0.07, "inches"), color = "gray50") +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  ggtitle("Bigram Network Graph") +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5))


total_transitions <- bigram_counts %>%
  group_by(syllable_sequence) %>%
  summarise(total = sum(n))


transition_probabilities <- bigram_counts %>%
  left_join(total_transitions, by = "syllable_sequence") %>%
  mutate(probability = n / total)

ggplot(transition_probabilities, aes(x = syllable_sequence, y = next_syllable, fill = probability)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "steelblue") +
  theme_minimal() +
  labs(title = "Bigram Transition Probabilities", x = "Syllable", y = "Next Syllable", fill =
"Probability")


### rank-frequency distribution

syllable_freq_table <- cleaned_syldata %>%
  count(Syllables, sort = TRUE, name = "frequency") %>%
  mutate(
    total = sum(frequency),
    term_frequency = frequency / total,
    rank = rank(-term_frequency),
    log_frequency = log10(frequency),
    log_term_frequency = log10(term_frequency),
```

```
   syllable_length = nchar(Syllables),
   log_rank = log(rank)
  )


#weight:

wt1 <- 1/syllable_freq_table$rank

######## plot Yule

Yule_model <- lm(log_term_frequency ~ I(rank) + I(log_rank), data = syllable_freq_table)

summary(Yule_model)

SSE0_yule=sum(Yule_model$residuals**2)
SSE0
AIC_yule <- AIC(Yule_model)


Yule_model_wt1 <- lm(log_term_frequency ~ I(rank) + I(log_rank),
                     data = syllable_freq_table,
                     weights = wt1)


summary(Yule_model_wt1)



pred <- predict(Yule_model)
pred1 <- predict(Yule_model_wt1)
ix <- order(syllable_freq_table$log_rank)


plot(syllable_freq_table$log_rank, syllable_freq_table$log_term_frequency,
    xlab = "Rank (in log scale)",
    ylab = "Syllable's Frequency (in log scale)",
    main = "Log-Log Syllable's frequency vs. Rank",
    cex.lab = 1.2, cex.axis = 1.2, cex.main = 1.5, pch = 19)


lines(syllable_freq_table$log_rank[ix], pred[ix], col = 'red', lwd = 2)
lines(syllable_freq_table$log_rank[ix], pred1[ix], col = 'blue', lwd = 2)

coef_unwt <- round(coef(Yule_model), 5)
coef_wt1 <- round(coef(Yule_model_wt1), 5)


x_pos <- min(syllable_freq_table$log_rank)
y_pos <- min(syllable_freq_table$log_term_frequency) - 0.1


text(x = x_pos, y = y_pos,
    labels = paste("Unweighted model:\nIntercept =", coef_unwt[1], "\nCoefficient 1 =", coef_unwt[2],
"\nCoefficient 2 =", coef_unwt[3],
                   "\nWeighted model w1:\nIntercept =", coef_wt1[1], "\nCoefficient 1 =", coef_wt1[2],
"\nCoefficient 2 =", coef_wt1[3]),
    adj = c(0,0), cex = 0.8, col = "black")
```

```
#### Zipf - power law


Zipf_model <- lm(log_term_frequency ~ I(log_rank), data = syllable_freq_table)

summary(Zipf_model)

SSE0_zipf=sum(Zipf_model$residuals**2)
AIC_zipf <- AIC(Zipf_model)
AIC_zipf


Zipf_model_wt1 <- lm(log_term_frequency ~ I(log_rank),
                     data = syllable_freq_table,
                     weights = wt1)
summary(Zipf_model_wt1)



pred <- predict(Zipf_model)
pred1 <- predict(Zipf_model_wt1)
ix <- order(syllable_freq_table$log_rank)


plot(syllable_freq_table$log_rank, syllable_freq_table$log_term_frequency,
    xlab = "Rank (in log scale)",
    ylab = "Syllable's Frequency (in log scale)",
    main = "Log-Log Syllable's frequency vs. Rank",
    cex.lab = 1.2, cex.axis = 1.2, cex.main = 1.5)


lines(syllable_freq_table$log_rank[ix], pred1[ix], col = 'blue', lwd = 2)
lines(syllable_freq_table$log_rank[ix], pred[ix], col = 'red', lwd = 2)


coef_unwt <- round(coef(Zipf_model), 3)
coef_wt1 <- round(coef(Zipf_model_wt1), 3)


x_pos <- min(syllable_freq_table$log_rank)
y_pos <- min(syllable_freq_table$log_term_frequency) - 0.1


text(x = x_pos, y = y_pos,
    labels = paste("Unweighted model:\nIntercept =", coef_unwt[1], "\nSlope =", coef_unwt[2],
                   "\n\nWeighted model w1:\nIntercept =", coef_wt1[1], "\nSlope =", coef_wt1[2]),
    adj = c(0,0), cex = 0.8, col = "black")


### Beta


Beta_model <- lm(log_term_frequency ~ I(log_rank) + log(max(rank) + 1 - rank), data =
syllable_freq_table)
```

```
summary(Beta_model)

SSE0_beta=sum(Beta_model$residuals**2)
SSE0_beta
AIC_beta <- AIC(Beta_model)
AIC_beta


Beta_model_wt1 <- lm(log_term_frequency ~ I(log_rank) + log(max(rank) + 1 - rank),
                     data = syllable_freq_table,
                     weights = wt1)


summary(Beta_model_wt1)

pred <- predict(Beta_model)
pred1 <- predict(Beta_model_wt1)
ix <- order(syllable_freq_table$log_rank)


plot(syllable_freq_table$log_rank, syllable_freq_table$log_term_frequency,
    xlab = "Rank (in log scale)",
    ylab = "Syllable's Frequency (in log scale)",
    main = "Log-Log Syllable's frequency vs. Rank")



lines(syllable_freq_table$log_rank[ix], pred1[ix], col = 'blue', lwd = 2)
lines(syllable_freq_table$log_rank[ix], pred[ix], col = 'red', lwd = 2)


coef_unwt <- round(coef(Beta_model), 3)

coef_wt1 <- round(coef(Beta_model_wt1), 3)



x_pos <- min(syllable_freq_table$log_rank)
y_pos <- min(syllable_freq_table$log_term_frequency) - 0.1


text(x = x_pos, y = y_pos,
    labels = paste("Unweighted model:\nIntercept =", coef_unwt[1], "\nCoefficient 1 =", coef_unwt[2],
"\nCoefficient 2 =", coef_unwt[3],
                   "\nWeighted model w1:\nIntercept =", coef_wt1[1], "\nCoefficient 1 =", coef_wt1[2],
"\nCoefficient 2 =", coef_wt1[3]),
    adj = c(0,0), cex = 0.8, col = "black")



####### Zipf Mandelbrot

ZM_model <- nls(log(term_frequency) ~ log(c_param) - a * log(rank + b),
                data = syllable_freq_table,
                start = list(a = 1, b = 0, c_param = 1),
                control = nls.control(maxiter = 100))
```

```
summary(ZM_model)

AIC_ZM <- AIC(ZM_model)
AIC_ZM
residuals_ZM <- residuals(ZM_model)
SSE0_ZM <- sum(residuals_ZM^2)
SSE0_ZM


ZM_modelw1 <- nls(log(term_frequency) ~ log(c) - a * log(rank + b),
                  data = syllable_freq_table,
                  start = list(a = 1, b = 0, c = 1),
                  control = nls.control(maxiter = 100),
                  weights = wt1)


ZM_modelw1



#### plots:

plot(log(syllable_freq_table$rank), log(syllable_freq_table$term_frequency),
    xlab = "log(rank)",
    ylab = "log(frequency)",
    main = "Log-Log Plot of Word Frequency vs. Rank")


lines(log(syllable_freq_table$rank), predict(ZM_model), col = 2, lwd = 3)
lines(log(syllable_freq_table$rank), predict(ZM_modelw1), col = 3, lwd = 3)


coef_unwt <- round(coef(ZM_model), 3)
coef_wt1 <- round(coef(ZM_modelw1), 3)


x_pos <- min(syllable_freq_table$log_rank)
y_pos <- min(syllable_freq_table$log_term_frequency) - 9


text(x = x_pos, y = y_pos,
    labels = paste("Unweighted model:\nIntercept =", round(coef_unwt[1], 3),
                   "\nCoefficient 1 =", round(coef_unwt[2], 3),
                   "\nCoefficient 2 =", round(coef_unwt[3], 3),
                   "\n\nWeighted model w1:\nIntercept =", round(coef_wt1[1], 3),
                   "\nCoefficient 1 =", round(coef_wt1[2], 3),
                   "\nCoefficient 2 =", round(coef_wt1[3], 3)),
    adj = c(0, 0), cex = 0.8, col = "black")


#### fitting



weighted_residuals_ZM <- sqrt(wt1) * residuals(ZM_modelw1)
weighted_RSS_ZM <- sum(weighted_residuals_ZM^2)
log_term_freq <- log(syllable_freq_table$term_frequency)
TSS <- sum((log_term_freq - mean(log_term_freq))^2)
```

```
rsquared_ZM_wt <- 1 - (weighted_RSS_ZM / TSS)
rsquared_ZM_wt


#### Akaike for weighted regression


aic_weighted <- function(model, weights) {
  n <- length(residuals(model))
  k <- length(coef(model))
  weighted_residuals <- sqrt(weights) * residuals(model)
  weighted_RSS <- sum(weighted_residuals^2)

  aic_w <- n * log(weighted_RSS / n) + 2 * k
  return(aic_w)
}


names(weighted_model_list) <- c("Yule_model_wt1", "Zipf_model_wt1", "Beta_model_wt1",
"ZM_model_wt1")


aic_values <- sapply(weighted_model_list, function(model) aic_weighted(model, wt1))
aic_values


### SSE for weighted


SSE_weighted <- function(model, weights) {
  weighted_residuals <- sqrt(weights) * residuals(model)
  SSE_w <- sum(weighted_residuals^2)
  return(SSE_w)
}


weighted_model_list <- list(Yule_model_wt1, Zipf_model_wt1, Beta_model_wt1, ZM_modelw1)


sse_values <- sapply(weighted_model_list, function(model) SSE_weighted(model, wt1))
sse_values


### number of parameters

K_yule <- length(coef(Yule_model))
K_zipf <- length(coef(Zipf_model))
K_beta <- length(coef(Beta_model))
K_ZM <- length(coef(ZM_model))


#### delata AIC


min_AIC_unwt <- min(c(AIC_yule, AIC_zipf, AIC_beta, AIC_ZM))
```

```
delta_AIC_yule_unwt <- AIC_yule - min_AIC_unwt
delta_AIC_yule_unwt
delta_AIC_zipf_unwt <- AIC_zipf - min_AIC_unwt
delta_AIC_zipf_unwt
delta_AIC_beta_unwt <- AIC_beta - min_AIC_unwt
delta_AIC_beta_unwt
delta_AIC_ZM_unwt <- AIC_ZM - min_AIC_unwt
delta_AIC_ZM_unwt


min_AIC_wt <- min(aic_values)

delta_AIC_yule_wt <- aic_values["Yule_model_wt1"] - min_AIC_wt
delta_AIC_zipf_wt <- aic_values["Zipf_model_wt1"] - min_AIC_wt
delta_AIC_beta_wt <- aic_values["Beta_model_wt1"] - min_AIC_wt
delta_AIC_ZM_wt <- aic_values["ZM_model_wt1"] - min_AIC_wt


#beta coef
beta_coeffs <- coef(Beta_model)
c0_beta <- beta_coeffs["(Intercept)"]
c1_beta <- beta_coeffs["I(log_rank)"]
c2_beta <- beta_coeffs["log(max(rank) + 1 - rank)"]
C_beta <- exp(c0_beta)
a_beta <- -c1_beta
b_beta <- c2_beta


# yule coef
yule_coeffs <- coef(Yule_model)
c0_yule <- yule_coeffs["(Intercept)"]
c1_yule <- yule_coeffs["I(rank)"]
c1_yule
c2_yule <- yule_coeffs["I(log_rank)"]
C_yule <- exp(c0_yule)
a_yule <- -c2_yule
b_yule <- exp(c1_yule)


# zipf

zipf_coeffs <- coef(Zipf_model)
c0_zipf <- zipf_coeffs["(Intercept)"]
c1_zipf <- zipf_coeffs["I(log_rank)"]
C_zipf <- exp(c0_zipf)
a_zipf <- -c1_zipf


#ZM

zm_coeffs <- coef(ZM_model)
c_zm <- zm_coeffs["c_param"]
a_zm <- zm_coeffs["a"]
b_zm <- zm_coeffs["b"]
```

```
results <- data.frame(
  Model = c("Beta", "Yule", "Zipf", "Zipf-Mandelbrot"),
  C = c(C_beta, C_yule, C_zipf, c_zm),
  a = c(a_beta, a_yule, a_zipf, a_zm),
  b = c(b_beta, b_yule, NA, b_zm)
)


##### Entropy

#### 1st part general entropy

distinct_syllables <- cleaned_sylldata %>%
  dplyr::select(Syllables, totalcount) %>%
  distinct()


distinct_syllables <- distinct_syllables %>%
  mutate(probability = totalcount / sum(totalcount))


distinct_syllables <- distinct_syllables %>%
  mutate(shannon_inf = -probability * log2(probability))


distinct_syllables <- distinct_syllables %>%
  mutate(rank = rank(-probability))


total_entropy <- sum(distinct_syllables$shannon_inf, na.rm = TRUE)

total_entropy


##### correcting sylldata according to cleaned_sylldata

unique_MSkiem <- unique(cleaned_sylldata$MSkiem)


filtered_sylldata <- sylldata %>%
  filter(MSkiem %in% unique_MSkiem)


mean_Tsyll <- mean(filtered_sylldata$Tsyll, na.rm = TRUE)

print(paste("The mean of Tsyll is:", mean_Tsyll))


sylldata_unnested <- filtered_sylldata %>%
  dplyr::select(moKur_nr, word, ZodIlg, stopai, Grupe, Tsyll, MSkiem, nr, autorID, autorius,
n_kuriniu, kurinioID, kurinys, autK1, autK, zanras, Versta, SK1:SK12) %>%
  pivot_longer(cols = starts_with("SK"), names_to = "Syllable_no", values_to = "Syllable") %>%
  filter(!is.na(Syllable))


syllable_counts <- sylldata_unnested %>%
  group_by(kurinys, zanras, Syllable) %>%
```

```
  summarize(count = n(), .groups = 'drop')


syllable_probs <- syllable_counts %>%
  group_by(kurinys, zanras) %>%
  mutate(total = sum(count), probability = count / total) %>%
  ungroup()


shannon_entropy_kurinys <- syllable_probs %>%
  group_by(kurinys, zanras) %>%
  summarize(entropy = -sum(probability * log2(probability)), .groups = 'drop')


word_counts <- syllldata_unnested %>%
  group_by(kurinys, zanras) %>%
  summarize(word_count = n(), .groups = 'drop')


entropy_with_wordcounts <- merge(shannon_entropy_kurinys, word_counts, by = "kurinys")



entropy_with_wordcounts$genre_english <- factor(entropy_with_wordcounts$zanras.x,
                                                levels = c("Apysaka", "Apsakymas", "Kelionės", "Mitologija", "Pasaka",
"Pjesė", "Poezija", "Romanas"),
                                                labels = c("Short story", "Essay", "Travel", "Mythology", "Tales",
"Plays", "Poetry", "Novel"))


ggplot(entropy_with_wordcounts, aes(x = word_count, y = entropy, color = genre_english)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Shannon Entropy vs Word Count by Book and Genre",
       x = "Word Count",
       y = "Shannon Entropy",
       color = "Genre")


#spearman correlation

spearman_test <- cor.test(entropy_with_wordcounts$word_count, entropy_with_wordcounts$entropy,
method = "spearman")

spearman_test



### shannon by genre

syllable_counts_genre <- syllldata_unnested %>%
  group_by(zanras, Syllable) %>%
  summarize(count = n(), .groups = 'drop')


syllable_probs <- syllable_counts_genre %>%
  group_by(zanras) %>%
  mutate(total = sum(count), probability = count / total) %>%
```

```
  ungroup()

shannon_entropy_zanras <- syllable_probs %>%
  group_by(zanras) %>%
  summarize(entropy = -sum(probability * log2(probability)), .groups = 'drop')

shannon_entropy_zanras

write.csv(shannon_entropy_kurinys, "shannon_entropy_kurinys.csv", row.names = TRUE)

word_counts <- syllldata_unnested %>%
  group_by(kurinys, zanras) %>%
  summarize(word_count = n(), .groups = 'drop')

entropy_with_wordcounts <- merge(shannon_entropy_kurinys, word_counts, by = "kurinys")

ggplot(entropy_with_wordcounts, aes(x = word_count, y = entropy, color = zanras.x)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Shannon Entropy vs Word Count by Book and Genre",
       x = "Word Count",
       y = "Shannon Entropy") +
  scale_color_discrete(name = "Genre")

### coonditional entropy

bigrams_list <- strsplit(filtered_syllldata$MSkiem, "-")

generate_bigrams_with_markers <- function(syllables) {
  if (length(syllables) == 1) {
    return(paste0("*_", syllables))
  }
  c(paste0("*_", syllables[1]),
    paste(head(syllables, -1), tail(syllables, -1), sep = "_"))
}
marked_bigrams_list <- lapply(bigrams_list, generate_bigrams_with_markers)

bigrams <- unlist(marked_bigrams_list)

context_freq <- table(sub("_.*", "", bigrams))

bigram_freq <- table(bigrams)
```

```
bigrams_df <- data.frame(bigram = names(bigram_freq), freq = as.numeric(bigram_freq))

p_c <- context_freq / sum(context_freq)


bigrams_df <- bigrams_df %>%
  separate(bigram, into = c("context", "syllable"), sep = "_", fill = "left", remove = FALSE) %>%
  mutate(context = ifelse(context == "*", NA, context))


bigrams_df <- bigrams_df %>%
  group_by(context) %>%
  mutate(p_xi_given_c = freq / sum(freq)) %>%
  ungroup()


bigrams_df$conditional_entropy <- ifelse(bigrams_df$p_xi_given_c > 0,
                                         -bigrams_df$p_xi_given_c * log2(bigrams_df$p_xi_given_c),
                                         0)


bigrams_df$context[is.na(bigrams_df$context)] <- "*"


bigrams_df$weighted_entropy <- bigrams_df$conditional_entropy *
p_c[as.character(bigrams_df$context)]

total_conditional_entropy <- sum(bigrams_df$weighted_entropy, na.rm = TRUE)


print(paste("Total Conditional Entropy:", total_conditional_entropy))


#### conditional entropy per genre, It is repeated for every genre. Here it is a part

#### for novel (romanas) genre

filtered_sylldata_romanas <- filtered_sylldata[filtered_sylldata$zanras == "Romanas", ]


bigrams_list_romanas <- strsplit(filtered_sylldata_romanas$MSkiem, "-")


generate_bigrams_with_markers <- function(syllables) {
  if (length(syllables) == 1) {
    return(paste0("*_", syllables))
  }
  c(paste0("*_", syllables[1]),
    paste(head(syllables, -1), tail(syllables, -1), sep = "_"))
}


marked_bigrams_list_romanas <- lapply(bigrams_list_romanas, generate_bigrams_with_markers)
```

```
bigrams_romanas <- unlist(marked_bigrams_list_romanas)


context_freq_romanas <- table(sub("_.*", "", bigrams_romanas))


bigram_freq_romanas <- table(bigrams_romanas)


bigrams_df_romanas <- data.frame(bigram = names(bigram_freq_romanas), freq =
as.numeric(bigram_freq_romanas))


p_c_romanas <- context_freq_romanas / sum(context_freq_romanas)


bigrams_df_romanas <- bigrams_df_romanas %>%
  separate(bigram, into = c("context", "syllable"), sep = "_", fill = "left", remove = FALSE) %>%
  mutate(context = ifelse(context == "*", NA, context))


bigrams_df_romanas <- bigrams_df_romanas %>%
  group_by(context) %>%
  mutate(p_xi_given_c = freq / sum(freq)) %>%
  ungroup()


bigrams_df_romanas$conditional_entropy <- ifelse(bigrams_df_romanas$p_xi_given_c > 0,
                                      -bigrams_df_romanas$p_xi_given_c *
log2(bigrams_df_romanas$p_xi_given_c),
                                      0)


bigrams_df_romanas$context[is.na(bigrams_df_romanas$context)] <- "*"


bigrams_df_romanas$weighted_entropy <- bigrams_df_romanas$conditional_entropy *
p_c_romanas[as.character(bigrams_df_romanas$context)]


total_conditional_entropy_romanas <- sum(bigrams_df_romanas$weighted_entropy, na.rm = TRUE)


print(paste("Conditional Entropy for Romanas:", total_conditional_entropy_romanas))


#### complex and not complex text classification


threshold <- quantile(shannon_entropy_kurinys$entropy, 0.75)
```

```
shannon_entropy_kurinys$Complexity <- ifelse(shannon_entropy_kurinys$entropy >= threshold, "Complex", "Not Complex")


merged_data <- merge(cleaned_sylldata, shannon_entropy_kurinys, by = "kurinys")
```

Classification of complex text in python code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import shuffle

np.random.seed(123)

fragment_size = 1000  #syllables per fragment
fragments = []

model_data = shuffle(model_data).reset_index(drop=True)

for name, group in model_data.groupby('kurinys'):
    for i in range(0, len(group), fragment_size):
        fragment = group.iloc[i:i + fragment_size]
        if len(fragment) == fragment_size:
            fragment_id = f"{name}_fragment_{i // fragment_size}"
            fragments.append((fragment_id, fragment))

aggregated_fragments = []

for fragment_id, fragment in fragments:
    position_proportions = fragment.groupby('Syll_position_group').size().div(len(fragment))
    type_proportions = fragment.groupby('Syll_type').size().div(len(fragment))
    stopai_proportions = fragment.groupby('stopai').size().div(len(fragment))
    cv_pattern_proportions = fragment.groupby('cv_pattern').size().div(len(fragment))

    complexity_class = fragment['Complexity'].iloc[0]

    aggregated_data = {
        'unique_syllable_count': fragment['Syllables'].nunique(),
        'avg_syllable_freq': fragment['n'].mean(),
        'cv_pattern_unique': fragment['cv_pattern'].nunique(),
        'Complexity': complexity_class,
        'fragment_id': fragment_id
```

```
    }

    aggregated_data.update({'position_' + k: v for k, v in position_proportions.to_dict().items()})
    aggregated_data.update({'type_' + k: v for k, v in type_proportions.to_dict().items()})
    aggregated_data.update({'stopai_' + k: v for k, v in stopai_proportions.to_dict().items()})
    aggregated_data.update({'cv_pattern_' + k: v for k, v in cv_pattern_proportions.to_dict().items()})

    aggregated_fragments.append(aggregated_data)

final_aggregated_data = pd.DataFrame(aggregated_fragments)

final_aggregated_data = final_aggregated_data.fillna(0)

X = final_aggregated_data.drop(['fragment_id', 'Complexity'], axis=1)
y = final_aggregated_data['Complexity']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#logistic regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
log_reg_pred = log_reg.predict(X_test)


#random forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

#KNN
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
#decision tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)

#GB
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)

#evaluation
models = [log_reg, rf, dt, gb, knn]
predictions = [log_reg_pred, rf_pred, dt_pred, gb_pred, knn_pred]
model_names = ['Logistic Regression', 'Random Forest', 'Decision Tree', 'Gradient Boosting', 'KNN']
```

```
for model, pred, name in zip(models, predictions, model_names):
    print(f"Model: {name}")
    print("Classification Report:")
    print(classification_report(y_test, pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, pred))
    print("-" * 50)

feature_importances = gb.feature_importances_

features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

sorted_features_df = features_df.sort_values(by='Importance', ascending=False)
print(sorted_features_df.head(10))
```