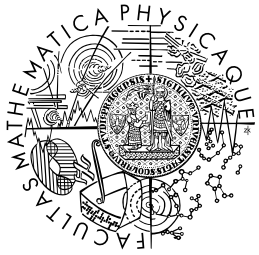

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Patrícia Březinová

Computational analysis and synthesis of song lyrics

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. Martin Popel, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Computational analysis and synthesis of song lyrics

Author: Patrícia Březinová

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Martin Popel, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Abstract.

Keywords: key words

Contents

Introduction

This works may include some literary or technical terms that the reader is not familiar with. For their definition, please see the chapter "Glossary of literary and technical terms" at the end of this thesis.

I can cite this Greene et al. [2010] when saying machine-generated poetry automatic evaluation is hard

Je otázka, zda bude vaše práce něĊím specifická pro písně (oproti básním), krom toho, že jako data máte písně. U písní by samozřejmě šlo dělat analýzu zvukové stránky (melodie, harmonie, rytmus) a studovat korelace s textovou stránkou, ale to je zřejmě mimo rámec této práce. Mohla byste to ale zmínit někde na zaĊátku, v zadání, že se schválně omezujete jen na ty (písňové) texty a ignorujete zvukovou stránku.

1. Related work

mention here rhyme generating tools like rhymezone

Is it OK that this chapter uses IPA but IPA is explained in the next sections?

uviest do related works aj 4 metody detekcie rymov od plechaca? ale vsetky priklady su machine learning okrem sparsaru

1.0.1 Rhyme types and literary devices

Although everyone instinctively knows what a rhyme is and can recognize one in a poem or a song, it does not have a very precise definition. It is described as "a word that has the same last sound as another word" by Cambridge Dictionary (Walter [2008]) or a "literary device, featured particularly in poetry, in which identical or similar concluding syllables in different words are repeated" by LiteraryDevices Editors [2020]. These definitions are not detailed enough to base a good algorithm off of, so let's look deeper into different rhyme types.

Perfect rhyme (also true rhyme, or sometimes just "rhyme") is the most common and valued type of rhyme. It requires two conditions to be met: cite?

- last stressed vowel and all following sounds are identical
- immediately preceding sounds differ

It is also the only rhyme for which the definitions are consistent (for example, see Bain [1867], van der Schelde [2020], Bergman [2017], [1]). It can be further distinguished depending on how many syllables are involved:

- **Masculine** (also single, monosyllabic) – "the commonest kind of rhyme, between single stressed syllables at the ends of verse" (Baldick [2008]). Examples:

  fly (flaɪ) / sky (skaɪ)

  before (bi-fɔr) / explore (ɪks-plɔr) [2] [3]

- **Feminine** (also double) – "a rhyme on two syllables, the first stressed and the second unstressed" (Baldick [2008]). Examples:

  bitten (bɪ-tən) / written (rɪ-tən)

  lazy (leɪ-zi) / crazy (kreɪ-zi)

- **Dactylic** (also triple) – "a rhyme on three syllables, the first stressed and the others unstressed"(Baldick [2008]). Examples:

  amorous (æ-mər-əs) / glamorous (glæ-mər-əs)

  vanity (væ-nɪ-ti) / humanity (ju-mæ-nɪ-ti))

[1] https://en.wikipedia.org/wiki/Rhyme

[2] For the examples, we are using IPA transcriptions because it is more comfortable for human reader.

[3] Stressed syllables are underlined. Syllables are separated with a hyphen.

Imperfect rhyme (also slant or half rhyme) rhymes "the stressed syllable of one word with the unstressed syllable of another word" (Bergman [2017]). Examples:

cabbage (kæ-bɪdʒ) / ridge (rɪdʒ)

painting (peɪ-nɪŋ) / ring (rɪŋ)

In other sources, definitions differ – for example LiteraryDevices Editors [2020] calls this effect "feminine rhyme". On the other hand, Baldick [2008] and The Editors of Encyclopaedia Britannica [2014] use the term "imperfect rhyme" for end-line consonance (see definition below) and van der Schelde [2020] uses it for end-line assonance (see definition below). For the purpose of this thesis we will work with the first definition mentioned.

Unaccented rhyme (also weakened rhyme) "occurs when the relevant syllable of the rhyming word is unstressed" (The Editors of Encyclopaedia Britannica [2014]). Examples:

hammer (hæ-mər) / carpenter (kɑr-pən-tər)

The difference opposed to imperfect rhyme is that here both rhyme fellows are unstressed.

Identical rhyme (also rime riche) is "a kind of rhyme in which the rhyming elements include matching consonants before the stressed vowel sounds." This includes "rhyming of two words with the same sound and sometimes the same spelling but different meanings e.g seen (sin)/ scene (sin). The term also covers word-endings where the consonant preceding the stressed vowel sound is the same: compare (kəm-pɛr) / despair (dɪ-spɛr)." (Baldick [2008]). It is generally considered not as good as perfect rhyme because it is too predictable for the listener[4].

Forced rhyme (also near rhyme) "includes words with a close but imperfect match in sound in the final syllables" Bergman [2017]. Examples:

green (grin) / fiend (find)

hide (haɪd) / mind (maɪnd)

This includes the case when spelling in changed in order to make the rhyme work, e.g. truth (truθ) / endu'th (en-duθ) (a contraction of "endureth"). It can also refer to using unnatural word order to get the rhyming word at the end of the line (Bergman [2017]) but we will not make use of this interpretation in this thesis.

Assonance is "repetition of stressed vowel sounds within words with different end consonants" (The Editors of Encyclopaedia Britannica [2014]). Examples:

quite (kwaɪt) / like (laɪk)

free (fri) / breeze (briz)

The term itself defines a literary device applicable anywhere in the poem but when used at the end of verse, it can sometimes be considered a rhyme (under various names) by sources like van der Schelde [2020], Bergman [2017] and others.

[4] https://literaryterms.net/rhyme/

Consonance is "the recurrence or repetition of identical or similar consonants" (The Editors of Encyclopaedia Britannica [2014]). Examples:

country (kən-tri) / contra (kɑn-trə)

hickory dickory dock (hɪ-kə-ri dɪ-kə-ri dɑk)

Similarly as assonance, it applies to repetition anywhere. When seen at the end of verse, it can be considered a rhyme and again, various terms are used, perhaps the most common is "pararhyme" (The Editors of Encyclopaedia Britannica [2014], Baldick [2008]).

The last two terms may seem as more of a tool for poets than songwriters. Surprisingly, they have found their way into song lyrics and have become a standard in genres like hip hop according to van der Schelde [2020]. From the creative point of view, it is not less sophisticated rather it enriches rhyme as we know it (Brogan [2016]).

Mention here we're not using it as a rhyme - or maybe if it will be highlighted in the visualization?

uviest to tam jasnejsie? definicie zo slovnikov nie su dost vyhranene (ex. consonance - nehoduju sa vowels!)

Other rhyme types exist e.g. eye rhyme where "the spellings of the rhyming elements match, but the sounds do not, e.g. love (ləv) / prove (pruv)" (Baldick [2008]). We will be omitting them because we did not consider them relevant for song lyrics or the purpose of this thesis.

spomenut ze nevieme zistit ci to spevak nevyslovi inak ,ze moze hltat bezprizvucne slabiky viac naraz

1.1 Rhyme detection tools

Pridat nejaky nastroj co pouziva len CMU dictionary

1.1.1 SPARSAR

1.1.2 Unsupervised approach

EM algorithm

Reddy and Knight [2011] proposed a language-independent model for finding rhyme schemes in poetry. They created an unsupervised model based on EM algorithm that assigns the most probable rhyme scheme for each sequence of line-final words. It achieved good results when tested on annotated English and French corpus with poetry from 15th to 20th century. However its big pitfall lies in the fact that it is biased towards the rhyme schemes from golden data. It has a predefined set of all rhyme schemes found in tested data and those are the only ones it chooses from. For illustration, in a 14-line stanza it can choose from 90 schemes which is only 0.00005% of all possible options. In 29% of cases from French corpus it has only one choice.[5]

[5] http://versologie.cz/talks/2017basel/

RhymeTagger

Plecháč [2018] came with a collocation-driven alternative named RhymeTagger. It uses the same dataset as the previous approach with addition of a larger Czech poetry corpus[6]. Each line-final word is transcribed into phonetic transcription and split into components – syllable peak for each syllable and consonant cluster in between. In the "expectation" step, probabilities for each component pair are calculated based on their co-occurrence in line-final words, e.g. conditional probability of rhyme based on peak component pair əʊːəʊ will be very high but for consonant component pair kːr quite low. These statistics for component pairs are then used in the "maximization" step to calculate the probability for line-final word pair as a combined probability of all their components (paired by means of usual association measure). If the probability of two words is above a given threshold they are considered a rhyme. After all such pairs are classified, probabilities are iteratively recalculated in the EM cycle.

For words that were not successfully classified with this method, there is a fallback. The author observed that some words are now pronounced differently than they were during the Shakespearean era they were written in and therefore using current pronunciation dictionaries may ruin the original rhyme, e.g. original near (nɛːr) / there (ðɛːr) vs. contemporary near (nɪːr) / there (ðɛːr). Original pronunciation can be therefore inferred from words with similar orthography. He calculated rhyme probability given final character trigrams, which helped achieve higher recall. Although other methods may have better precision, collocation-driven approach wins in recall as seen in 1.1. This method does not take into account meter.

For evaluation, they used precision, recall and F-score calculated as follows:

$$PRECISION = \frac{true\ positives}{true\ positives + false\ positives}$$

$$RECALL = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$F - SCORE = \frac{2 * PRECISION * RECALL}{PRECISION + RECALL}$$

For an intuitive view, the reader can imagine precision as how many of algorithm's rhymes were actually rhymes and recall as how many rhymes were discovered. F-score describes the trade-off between the two.

Popisat preco nejdu ich data pouzit na testovanie mojej detekcie rymov

Deep-speare

As a part of their sonnet quatrain generating model, Lau et al. [2018] have implemented a Rhyme component that identifies and generates rhymes. It is a unidirectional forward LSTM (Hochreiter and Schmidhuber [1997]) that learns to separate rhyming word pairs from non-rhyming. They generate input by pairing one line-final word with the other three from the same quatrain. Since the rhyme scheme of a sonnet quatrain is always *abab* this will result in one rhyming

[6] https://github.com/versotym/corpusCzechVerse

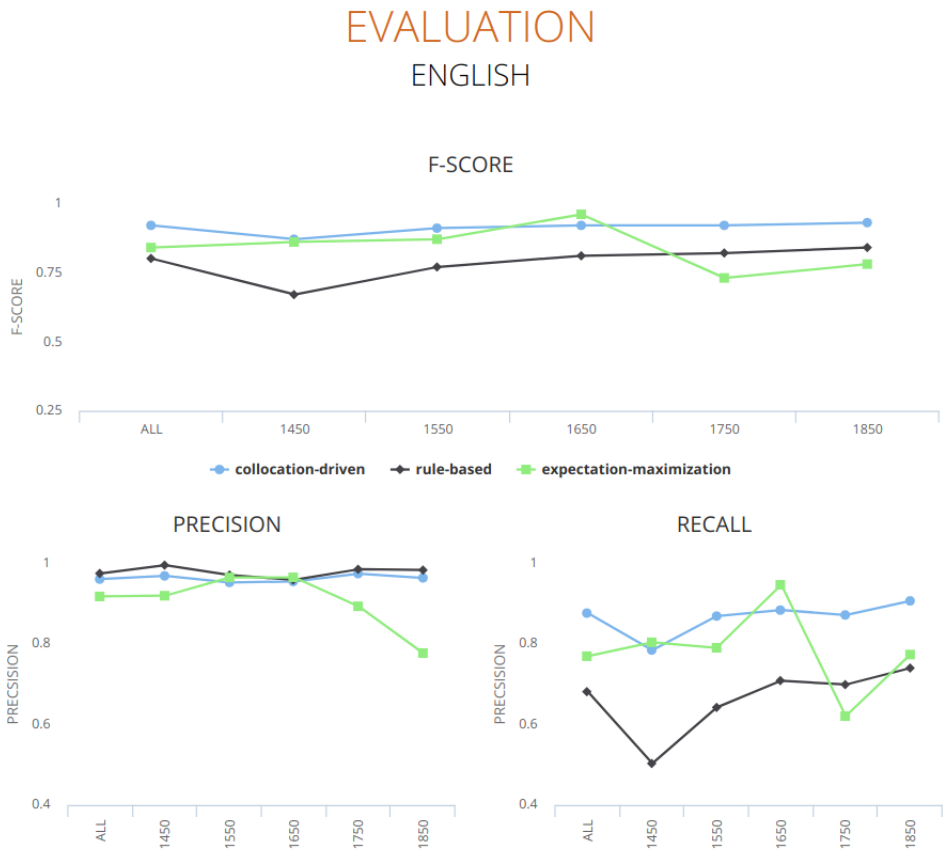

Figure 1.1: Evaluation of RhymeTagger on English corpus in comparison with EM algorithm and simple rule-based approach. The x axis is the year when the tested poem was written, the y axis are the evaluation scores as described above. Reproduced from Plecháč.

pair and two non-rhyming. Additional non-rhyming pairs are generated with random word sampling. Then the model with margin-based loss learns the margin separating the best pair from all the others. It returns a cosine similarity score that estimates how well do two words rhyme.

To evaluate this model, authors used phoneme matching with CMUdict [7] and the EM model from Reddy and Knight [2011] trained on their own data and they were able to outperform both based on F1 score.

1.2 Visualization tools

In the following section, we will describe existing visualization tools for poetry. Software mentioned below focuses on poems, however song lyrics can be considered just a more structurally relaxed version of a regular poem.

[7] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

1.2.1 Poem Viewer

Quite complex and comprehensive visualization tool is Poem Viewer Abdul-Rahman et al. [2013]. With no need for complicated installations it is easily available for the writers as a web-based application as shown in Figure 1.2. Unfortunately, at the time of writing this thesis the upload of custom text was not working. Luckily, this is still an ongoing project so this might be just a temporary issue. Nevertheless there are some default poems available to demonstrate this software's capabilities.

Figure 1.2: Screenshot from Poem Viewer tool – visualizing Love by Elizabeth Barrett Browning.

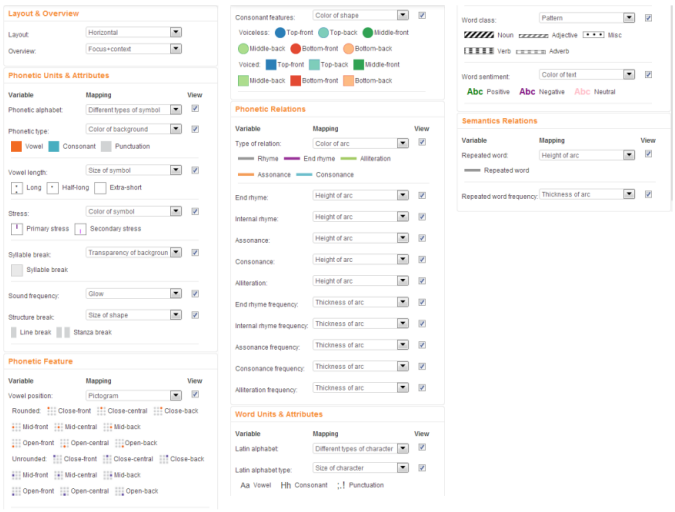

Figure 1.3: Available options and their default mappings in Poem Viewer.

Most of the analyzed features (shown in Figure 1.3 focus on the phonetic aspects of the poem. After phonetic transcription to IPA users can analyze consonant features, vowel length and position, stress, syllables, word classes and

sentiment using color codes and markers. A second layout offers six different graphs/animations of tongue positions during each verse. Arcs are used to mark end rhyme, alliteration, assonance, consonance, their particular frequencies and repeating words.

Overall this software, although very elaborate, appears crammed and confusing for an inexperienced user. Moreover, it is perhaps better suited for its original use case – a well-structured poem – than less regular song lyrics.

1.2.2 SPARSAR

SPARSAR (Delmonte and Prati [2014]) is also a very interesting tool for poetry analysis and expressive Text-to-speech conversion. It is originally designed for a thorough examination of a very strictly structured Shakespeare's sonnets. To achieve this, it has to run analyses on many levels – and these results can be used to analyze any poem. It looks at the poem on three levels: phonetic (pronunciation, consonant and vowel tongue position, assonances, etc.), poetic (metrical structure, rhyme schemes, acoustic length, etc.), and semantic (sentiment, methaphorically linked words, anaphora, etc.).

User can choose between a window application with graphs and diagrams or a headless mode with .xml output files. Its main disadvantage for our use case is that it is written in Prolog and therefore is very strict on the input format and runs only under a specific older version of Ubuntu.

1.2.3 ProseVis

This Java desktop visualization tool by Clement et al. [2013] analyzes text through parts-of-speech, phonemes, stress, tone, and break index. These features are extracted using OpenMary Text-to-speech system (Schröder et al. [2006]) and predictive classification. The authors believe their visualization will present the features to user in a more human readable form (ProseVis [2014]).

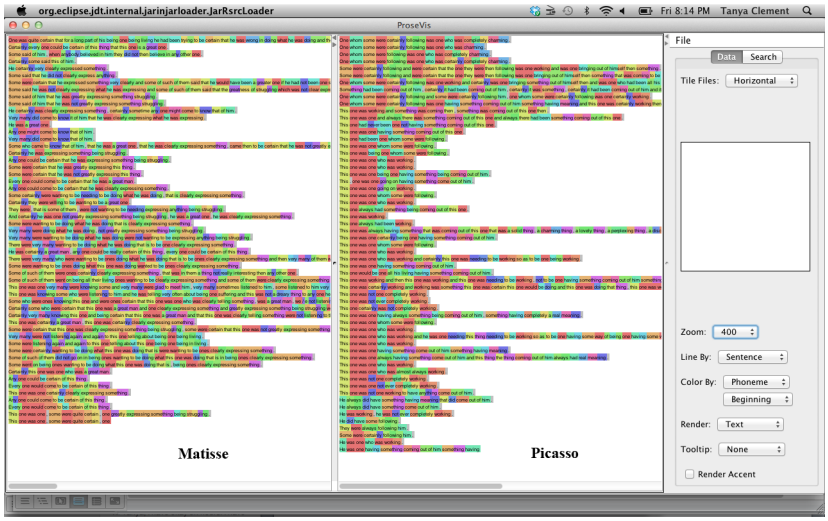

Figure 1.4: Comparison of two poems in ProseVis. Reproduced from ProseVis [2014].

1.2.4 Poemage and RhymeDesign

Poemage (McCurdy et al. [2015a]) and RhymeDesign (McCurdy et al. [2015b]) are both open-source applications with focus on analysis of sonic devices and sonic topology in poetry. Poemage[8] focuses on complex structures of words connected through some sonic or linguistic resemblance across the space of the poem. It is available for MacOS or Windows with a web version currently under development. In MacOS application RhymeDesign – which also provides the backend for Poemage – users can enter their poem and query for one of the default rhyme types or choose a custom rhyme type.

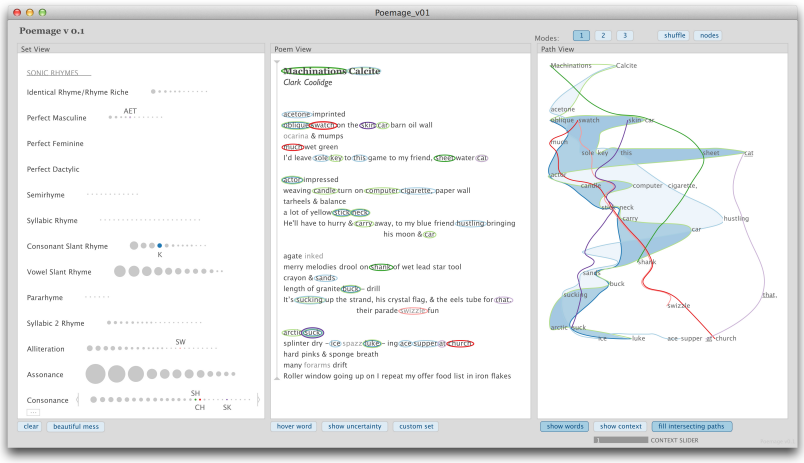

Figure 1.5: An example analysis in Poemage.

1.2.5 Ambiances

This software is unique in the fact that the analysis is integrated in the process of writing. As described in the paper Meneses and Furuta [2015], writers enter the poem, receive a visualization and can control this visualization with body and hand gestures which in turn influence the poem. By such interconnection the authors aim to make Ambiances a part of the writing process and give it a chance to influence the final result. However, the actual software does not seem to be openly available.

1.3 Generation tools

Lau et al. [2018] - learns rhyme automatically, for sonnets, results indistinguishable, apart from expert evaluation, missing emotion

generation by Reddy

[8] http://www.sci.utah.edu/~nmccurdy/Poemage/

2. Data

A crucial part of every analysis are the data. To be able to conduct an analysis with results that can reasonably represent the domain, we need to have enough of them - the more the better. Our dataset consist of 658,460 song lyrics scraped from the crowd-sourced website Genius[1]. Sadly, the original author of the dataset is unknown, it has been passed on to us by a colleague as a potentially interesting source for research. However all song lyrics are publicly available on the Genius website.

Not sure how to articulate this not to sound that bad...

2.1 Preprocessing

In most areas it is very hard to find a dataset of good quality and large quantity. Usually at least one of the two suffers. It is not any different with our data - although the dataset is large, the contents were created by ordinary people and intended for human readers so they are not well suited for automated processing. It is necessary to look closely at the data, remove faulty or redundant items, and clean the rest with preprocessing.

We received the dataset in JSON format, with each song as a separate item, each containing following features:

- *title* - the name of the song
- *lyrics* - the text of the song's lyrics
- *album* - song's album (or null)
- *genre* - one of the following: rap, pop, rock, r-b, country
- *artist* - song's performer
- *url* - the url of the lyrics page on Genius website
- *year* - the year the song was produced
- *is_music* - boolean flag distinguishing song lyrics from other texts
- other song details: *producer, featured artist, recording location, charts, writer, samples, sampled in, has featured video, has featured annotation*
- other website specific information: *rg artist id, rg type, rg tag id, rg song id, rg album id, rg created, has verified callout*

The features from the last two points were *null* for all or most of the items. That, and the fact that they do not give us much more information that would contribute to the lyrics analysis, made them useless and so we decided not to keep them. We removed all songs for which the attribute *is_music* was False,

[1] https://genius.com/

indicating it was not a song (often a poem or prose) or they did not contain lyrics - 34,259 songs in total. We further removed one song with invalid (incomplete) JSON. After comparing the lyrics to each other, we found and removed 32,551 duplicates.

Upon further inspection, we found out that our dataset also contains lyrics in different languages. We used the neural network model for language identification by Google (CLD3)[2] for the classification. It showed that our dataset contains exactly 100 different languages, most of them represented only marginally. Since other languages did not have enough data to support a good analysis and implementing them would be above the scope of this thesis, we kept only English lyrics. We further removed 832 items with language detection errors. All of them were under 10 lines long so they would not be a valuable addition anyway. At this point our dataset contained 438,037 items.

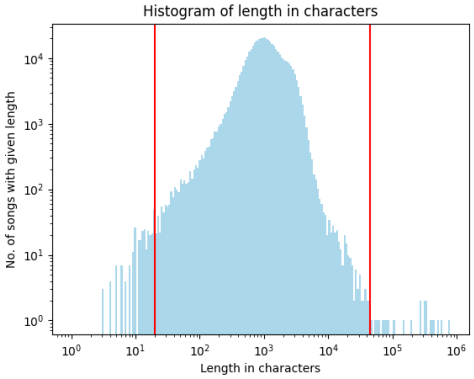

Figure 2.1: Histogram of number of characters in songs of our dataset.

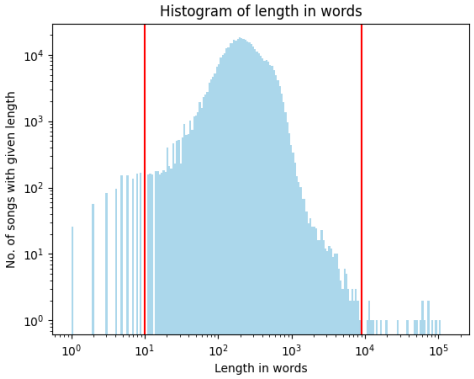

Figure 2.2: Histogram of number of words in songs of our dataset.

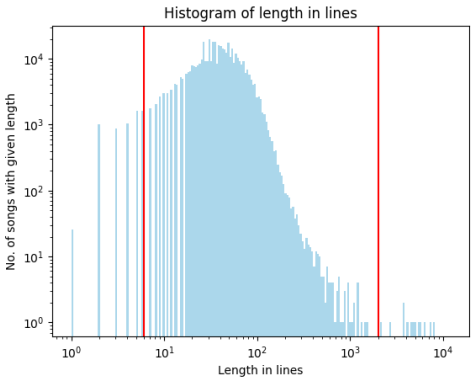

Figure 2.3: Histogram of number of lines in songs of our dataset.

To learn more about our data, we created histograms with song's length in characters, words, and lines (see Figures 2.1, 2.2, 2.3). Knowing the common issues of extreme values, we manually examined a few of the shortest and a few of the longest items. Confirming our expectations, we found out that they

[2] https://github.com/google/cld3

were not valid songs either. The long ones were usually book excerpts or rap improvisation battles, while the short ones were often links to advertisements or motivational quotes. We removed 14 long and 1,838 short lyrics. Although it may not seem as much, it could have strong negative influence mainly during the generation phase. In the Figures 2.1, 2.2, and 2.3, red lines mark the borders for removal - only the items in between them were kept.

Should I move the boundaries to remove more extremes?

2.2 Structure of the data

This section gives statistical information about the dataset after preprocessing. Table 2.1 sums up basic statistics about the data. All the attributes are listed in Table 2.2 along with the number of items for which these attributes have non-empty values. Lastly, pie chart in Figure 2.4 shows the portions of the data belonging to each genre.

Total number of songs:	436,185
Average number of lines per song:	43.3619
Average number of words per line:	5.9574

Table 2.1: Basic statistics about the dataset.

Attribute	Non-empty values
lyrics	436,185
title	436,179
album	112,060
genre	436,185
artist	436,184
url	436,185
year	96,491
lang	436,185
id	436,185
word_count	436,185

Table 2.2: Attributes and their counts of non-empty values.

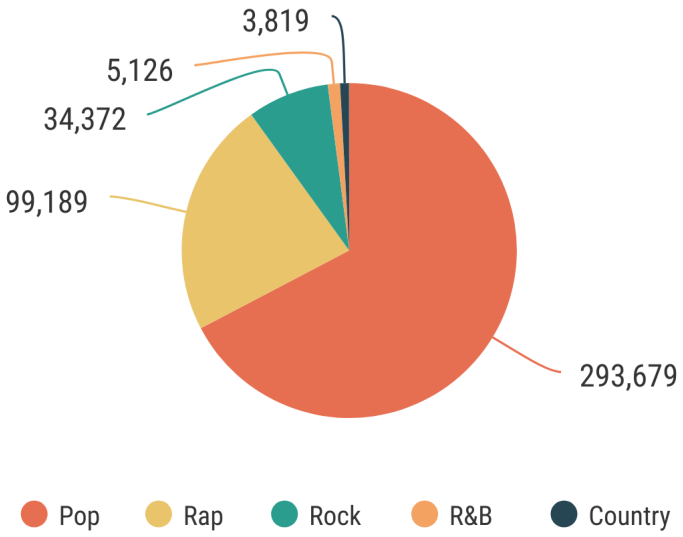

Figure 2.4: Distribution of genres in the dataset.

3. Lyrics evaluation

sucast zadania ze to bude bez zvukovej stranky hoci to vplyv ma

3.1 Rhyme detection

Describe the original attempt with SPARSAR.

3.1.1 Pronunciation

Unlike many other languages, English does not have a straightforward pronunciation rules. Therefore to be able to assess rhymes, we need to transcribe our text into a phonetic alphabet first. There are two commonly used alphabets to choose from – IPA and ARPAbet. The original International Phonetic Alphabet (IPA) used since 1888 uses one UNICODE character to encode each phoneme and it is commonly used for example in dictionaries. Since it uses non-ASCII characters, ARPAbet was developed as an equivalent for computers. It has two versions: 1-character that uses upper-case and lower-case letters and 2-character version where each phoneme is represented by one or more upper-case ASCII characters (**?**)(see Table3.1 for comparison). We will be using the 2-character ARPAbet because it is used by the CMUdict.

Example word	IPA	1-character ARPAbet	2-character ARPAbet
st**o**ry	ɔ	c	AO
bu**tt**er	ɾ	F	DX

Table 3.1: Comparison of different pronunciation alphabets.

Carnegie Mellon University Pronouncing Dictionary (CMUdict) is an open-source pronunciation dictionary.[1] Currently it contains 134,373 words (including their inflections) and their pronunciations in 2-character ARPAbet. For each word, there is one or several possible pronunciations in North American English including stress markers for primary, secondary or no stress. For the implementation, we used its Python wrapper package *cmudict* [2]. To use this we need to strip the input of punctuation and convert it to lower case.

CMUdict is a large dictionary and it includes also slang words so it should cover most of our input. To test this, we looked at all last words on each line of our data (since those are the important ones for rhyme analysis) and we found out that 5.52% of them are not in CMU dictionary. These included:

- uncommon words, e.g. superglue, redundantly
- misspelled words, e.g. decsion, girlfren
- numbers

[1] http://www.speech.cs.cmu.edu/cgi-bin/cmudict
[2] https://pypi.org/project/cmudict/

- foreign words, e.g. revoluccion, ecolli
- interjections and onomatopoeia, e.g. shoooshooo, woahwoah

Also, we applied some further data preprocessing that ensured more words in data would be found in the dictionary. We replaced the closing quotation mark "’" with the typewriter apostrophe "'" since only the second variant of apostrophe is accepted by CMUdict. We replaced hyphen "-" with a space " " to separate the hyphen-connected compound words into individual component words that have a higher chance of being found in the dictionary than the full version.
Describe here how we dealt with the ones not in CMUdict. Open MaryTTS?

3.1.2 Syllabification

Once we have the pronunciations, we can start to compare them. When comparing lines for rhymes, we have to establish a system of alignment so that we analyze only relevant pairs of phonemes. Initially, we created a simple rhyme detector that just traversed both verses backwards phoneme by phoneme and compared them. However, rhyming words do not have to have an equal number of phonemes. For example words in the Table 3.2 have a 2-syllable rhyme. However if we compared each phonemes one by one they get misaligned on consonant clusters S-T-R and P-L and we will miss the second syllable rhyme.

Word	ARPAbet transcription
constrain	K AH N - S T R EY N
complain	K AH M - P L EY N

Table 3.2: Example of misalignment when aligning by phonemes.

We need to make sure that we are comparing corresponding parts of verses otherwise we will miss the rhyme. A better approach would be to compare corresponding syllables. Each syllable can be further split into 3 groups ("CVC") – leading consonant cluster, vowel, and trailing consonant cluster. Consonant clusters can sometimes be empty. For syllabification we used python library *syllabify* [3] which conveniently returns syllables in CVC triplets as described above.

3.1.3 Calculating rating for one rhyme

Finally, we have extracted pronunciation and syllables so we can continue to analyze the rhyme and rate the song. We decided to first calculate ratings for pairs of verses and then create an overall song rating based on these individual ratings. Another approach would be to analyze the complex statistics of the entire song and rate it at the end all at once. The second method could be better at incorporating the high-level properties like repeating of the refrain. We chose the first approach because it is more straight-forward and gives us a number for each rhyme which can be more interesting for the writer. Additional high-level analysis can be added later if necessary.

[3] https://github.com/kylebgorman/syllabify

So let's focus on the rhyme analysis of two verses – or rhyme fellows – as they are typically called. Rhymes are located at the end of each line so there is no need to analyze the entire verse. How far should we look? The first choice would be to look at the last word. However rhymes can extend over more words as we see in

Find an example of multi-word rhyme where the second word is unaccented

. When we look at the rhyme types, the basic ones do not go further then the first stressed syllable (looking at the line backwards). Notably, even if the rhyme does extend further we can ignore the rest because it will not contribute to the rating. According to our research, the most perfect rhyme is perfect rhyme so it should get the perfect score. And if there are more rhyming syllables preceding the perfect rhyme, they cannot make the score better. Similarly, if the rhyme is not perfect, syllables preceding the final stress would already be considered an internal rhyme – which is also used (mainly in rap lyrics) but less valued than the classical end rhyme. We will therefore limit our window to the minimum number of syllables needed to include the stressed syllable in both rhyme fellows. Having a sequence of four unstressed syllables is very unlikely in English language so we limited our word preprocessing (pronunciation + syllabification) to last 4 syllables to speed up the performance.

To determine the rhyme we need to assess the match in sound of individual phonemes. Since we have each syllable separated into three groups we decided to give each group a number between -1 and 1 that would represent how similarly they sound. The numbers are assigned according to following heuristic:

Rewrite this list in a better-readable manner.

- 1 – all phonemes are identical
- 0.75
  - one is a subset of another
  - it is a pair of similar sounding phonemes
- A number between 0 and 1 if both are consonant clusters. This number is calculated:
  1. Add 1/(length of the shorter consonant cluster) for each **shared** sound at the beginning or the end of the consonant clusters.
  2. Add 0.75/(length of the shorter consonant cluster) for each **similar** sound at the beginning or the end of the consonant clusters.
- 0 – both are empty
- -0.5 – one of them is empty
- -1 – otherwise, meaning no matching or similar sounds

For an example of similarity evaluation see Table 3.3.

To identify similar sounds, we look them up if there is a similarity group containing both of them. These similarity groups were created...

1^{st} verse	on	her	front	door
2^{nd} verse	can't	stand	no	more
1^{st} pronunciation	_, AA, N	HH, ER, _	F R, AH, N T	D, AO, R
2^{nd} pronunciation	DH, AH, _	P, EY, N	N, OW, _	M, AO, R
similarity	-0.5, 0.75, -0.5	-1, -1, -0.5	-1, -1, -0.5	-1, 1, 1

Table 3.3: Example similarity evaluation for the last 4 syllables of two verses from song Cheatin' Woman by Lynyrd Skynyrd.

Describe how it was done. The iterative appraoch? Or maybe Holtman's hierarchy? panPhon? Mortensen et al. [2016]

Words for which we have not found pronunciation cannot be further processed so they are given rhyme rating 0 and skipped. Some words may have multiple possible pronunciations – in that case we evaluate each possible combination of pronunciations for given line pair. After we assign a rating for each combination, we will keep only the best rated combination of pronunciations and discard the rest.

With these similarity values we can proceed to calculate the rating which we decided to be as typically between 0 and 1. We will look at it syllable by syllable a return an average. There are some cases we need to consider individually:

- different – if all similarities are equal to -1, we can definitely say they do not rhyme and return a rating of 0
- identical – since identity is a weaker rhyme than perfect, it will be given a penalty for "little creativity" returning a fixed rating of 0.8
- perfect – perfect rhyme has a specific structure and if that holds, we can return the perfect score of 1.0

For the remaining cases we will create rules based on how rhymes behave. Not all phonemes are equally important so let's assign weights to reflect it. The key role in rhyming plays the vowel so it should have the strongest impact on the rating. Second important is the ending consonant because it is closer to the end. Beginning consonant can add up to a nicer rhyme but it cannot bear the rhyme on its own. Since the vowel itself can be enough to create the rhyming effect it should have more weight than the rest combined. Therefore we assigned weights as follows:

Beginning consonants: 1, Vowel: 4, Ending consonants: 2

The rating for one syllable is created as normalized sum of weights times similarities. Furthermore, we need to account for stress. We can do that by multiplying the result with a multiplication factor depending on how does the stress match.

- 1.0 for stressed rhyme because it is the strongest
- 0.9 for unstressed rhyme – it is weaker but the stress pattern matches
- 0.8 for an mismatching stress pattern

The final formula for a rhyme rating is an average of syllable ratings and looks like this:

$$average(stress\_multiplication\_factor * weighted\_average(similarities))$$

This formula formatting looks weird but I don't have an idea how to do it better.

3.1.4 Calculating song rating

The next step is to combine these rhyme ratings into one final rating for the entire song. In the previous section we created a function that returns rhyme rating for given two verses. To search for rhymes in the full lyrics we need to decide which verse pairs to check. The most straight-forward approach would be "brute force" – try each line with all the other lines. Besides its obvious disadvantage of increased time requirements it also detects rhymes that span across tens of lines. It is not strictly defined how many lines apart can the rhyme fellows be to still be considered a rhyme – the author can even make it a part of his artistic expression e.g. in "Author's Prologue" by Thomas [1952] the 1st line rhymes with 102th, 2nd with 101th and so on. But realistically, a rhyme between a line at the beginning of the lyrics and 20 lines later would not have an effect on the song listener – it requires a close proximity of rhyme fellows within the poem. Since the most common stanzaic form in English is a quatrain, a stanza of four lines (Eastman et al. [1970]), we decided to set the distance to 3. Further than that the effect on the reader gets weaker really fast.

This probably needs a citation but I can't find any.

As we decided, for each line, we will look at 3 preceding lines to look for its rhyme fellow. In case multiple of them rhyme, the one with the smallest distance will be selected – if it is closer it is more probable the reader will associate it with the rhyme in the next line because it is the strongest in the memory. Consequently, a rating representing the score of this rhyming pair will be saved for the second rhyme fellow. It will not be saved for the first to ensure it will not be added to the final rating twice. The first of the pair will either keep a rating it shares with another line before or it will get an "X" (as seen in Table 3.4) that represent no rating.

Let's focus now on the first four line marked with rhyme scheme letter "e" in Table 3.4. They all rhyme with each other and all rhymes except for *are/dark* are perfect. That means the line with "dark" would receive less than perfect score and it would lower the score of the entire song. If we instead ignore this rhyme and associate only the first two and the last two rhymes together we will receive a perfect score. Loosing rating by marking weaker rhymes does not make sense so we must add an exception to only keep the better score.

Using the ratings added to the lines a final score will be calculated as the average of lines that have a rating (not an "X").

Counterexample - we have a 4-line song where 2 lines rhyme and a 24-line song where 2 lines rhyme - they will receive the same score. That does not seem fair?

Rhymes in songs or poems are typically marked using a rhyme scheme. That means each verse gets assigned a letter – lines that share the same letter rhyme and those with different letters do not. We also decided to adapt this common notation. In case the song needs more letters than there are in the alphabet we will add another letter and continue alphabetically – aa, ab, ac, ..., ba, bb, bc, ..., ca, etc.

Rhyme	Rating	Lyrics	Pronunciation (last 2 syllables)
a	X	Twinkle twinkle little star	'T AH L', 'S T AA R'
a	1.0	How I wonder what you are	'Y UW ', ' AA R'
b	X	Up above the world so high	'S OW ', 'HH AY '
b	1.0	Like a diamond in the sky	'DH AH ', 'S K AY '
a	1.0	Twinkle twinkle little star	'T AH L', 'S T AA R'
a	1.0	How I wonder what you are	'Y UW ', ' AA R'
c	X	When the blazing sun is gone	' IH Z', 'G AO N'
c	0.8	When he nothing shines upon	' AH ', 'P AA N'
d	X	Then you show your little light	'T AH L', 'L AY T'
d	1.0	Twinkle twinkle all the night	'DH AH ', 'N AY T'
e	X	Twinkle twinkle little star	'T AH L', 'S T AA R'
e	1.0	How I wonder what you are	'Y UW ', ' AA R'
e	X	Then the traveler in the dark	'DH AH ', 'D AA R K'
e	1.0	Thanks you for your tiny spark	'N IY ', 'S P AA R K'
f	X	He could not see which way to go	'T UW ', 'G OW '
f	1.0	If you did not twinkle so	'K AH L', 'S OW '
e	X	Twinkle twinkle little star	'T AH L', 'S T AA R'
e	1.0	How I wonder what you are	'Y UW ', ' AA R'
g	X	In the dark blue sky you keep	'Y UW ', 'K IY P'
g	1.0	Often through my curtains peep	'T AH N Z', 'P IY P'
h	X	For you never shut your eye	'Y AO R', ' AY '
h	1.0	Till the sun is in the sky	'DH AH ', 'S K AY '
i	X	Twinkle twinkle little star	'T AH L', 'S T AA R'
i	1.0	How I wonder what you are	'Y UW ', ' AA R'

Table 3.4: Example of song analysis of the nursery rhyme "Twinkle, Twinkle Little Star" with a final rating of 0.989.

Since meter plays an important role in rhymes, another relevant property to examine is the number of syllables. To count syllables for each line we used Python package *syllabify* [4] which returns syllables using ARPAbet transcription. For words not in CMUdict we used a simple heuristic – since the nucleus of each syllable is most often a vowel (except for syllabic consonants) we counted the number of (groups of) vowels and used it as an estimation for the number of syllables. Although this gives a wrong estimate for words like *rhythm* or *house*it performed quite well when we tested it on a few out-of-dictionary words from our dataset. We found it unnecessary to try to further improve the heuristic since

[4] https://github.com/kylebgorman/syllabify

the words that are not in CMUdict are often foreign words that do not follow the standard pronunciation rules of English so any application of these rules would probably be of little help.

Nemalo by tu mozno byt este nieco o tom ako sme to testovali? Resp. netreba to odtestovat nejak systematickejsie nez od oka?

Conclusion

Bibliography

A. Abdul-Rahman, J. Lein, K. Coles, E. Maguire, M. Meyer, M. Wynne, C.R. Johnson, A. Trefethen, and M. Chen. Rule-based visual mappings – with a case study on poetry visualization. *Computer Graphics Forum*, 32(3):381–390, 2013. doi: 10.1111/cgf.12125. URL http://dx.doi.org/10.1111/cgf.12125.

Alexander Bain. *English composition and rhetoric, a manual*. New York, Appleton, 1867.

Chris Baldick. *The Oxford Dictionary of Literary Terms*, volume 3. Oxford University PressPrint, 2008. ISBN 9780199208272.

Bennet Bergman. Rhyme, 2017. URL https://www.litcharts.com/literary-devices-and-terms/rhyme.

T. V. F. Brogan. *The Princeton Handbook of Poetic Terms*, volume 3. Princeton University Press, 2016. ISBN 9781400880645.

Tanya Clement, David Tcheng, Loretta Auvil, Boris Capitanu, and Megan Monroe. Sounding for meaning: Using theories of knowledge representation to analyze aural patterns in texts. *DHQ: Digital Humanities Quarterly*, 7(1), 2013.

R. Delmonte and A. M. Prati. SPARSAR: An expressive poetry reader. pages 73–76, apr 2014. doi: 10.3115/v1/E14-2019. URL https://www.aclweb.org/anthology/E14-2019.

Arthur M Eastman, Alexander Ward Allison, Herbert Barrows, et al. *The Norton Anthology of Poetry*. Norton New York, 1970.

Erica Greene, Tugba Bodrumlu, and Kevin Knight. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 524–533, 2010.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. Deep-speare: A joint neural model of poetic language, meter and rhyme. *arXiv preprint arXiv:1807.03491*, 2018.

LiteraryDevices Editors. Rhyme - examples and definition of rhyme as a literary device, Dec 2020. URL https://literarydevices.net/rhyme/.

Nina McCurdy, Julie Lein, Katharine Coles, and Miriah Meyer. Poemage: Visualizing the sonic topology of a poem. *IEEE transactions on visualization and computer graphics*, 22(1):439–448, 2015a.

Nina McCurdy, Vivek Srikumar, and Miriah Meyer. Rhymedesign: A tool for analyzing sonic devices in poetry. In *Proceedings of the Fourth Workshop on Computational Linguistics for Literature*, pages 12–22, 2015b.

Luis Meneses and Richard Furuta. Visualizing poetry: Tools for critical analysis. *paj: The Journal of the Initiative for Digital Humanities, Media, and Culture*, 3, 2015.

David R. Mortensen, Patrick Littell, Akash Bharadwaj, Kartik Goyal, Chris Dyer, and Lori S. Levin. Panphon: A resource for mapping IPA segments to articulatory feature vectors. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3475–3484. ACL, 2016.

Petr Plecháč. A collocation-driven method of discovering rhymes (in czech, english, and french poetry). In *Taming the Corpus*, pages 79–95. Springer, 2018.

Petr Plecháč. Collocation-driven method of discovering rhymes in a corpus of czech, english, and french poetic texts. URL http://versologie.cz/talks/2017basel/.

ProseVis. Prosevis, 2014. URL https://sourceforge.net/projects/prosevis/.

Sravana Reddy and Kevin Knight. Unsupervised discovery of rhyme schemes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 77–82, 2011.

Marc Schröder, Anna Hunecke, and Sacha Krstulovic. Openmary–open source unit selection as the basis for research on expressive synthesis. In *Blizzard Workshop*, 2006.

The Editors of Encyclopaedia Britannica. Encyclopedia britannica, 2014. URL https://www.britannica.com/.

Dylan Thomas. Author's prologue. *Collected Poems 1934-1952*, 1952.

Jona van der Schelde. Phonological and phonetic similarity as underlying principles of imperfect rhyme. 2020.

Elizabeth Walter. *Cambridge advanced learner's dictionary*. Cambridge university press, 2008.

List of Figures

List of Tables

Glossary of literary and technical terms

consonant cluster A sequence of syllables without a vowel. 3

headless mode A mode in which software runs on hardware without a graphic user interface, e.g. a script in terminal. 6

LSTM Long-Sort Term Memory - a type of recurrent neural network. 5

quatrain A type of stanza consisting of four lines. 5

sonnet A poetic form traditionally containing 14 lines written in iambic pentameter with rhyme scheme *abab cdcd efef gg*. 5

syllable peak A nucleus of a syllable - either a vowel or a syllabic consonant. 3

A. Attachments

A.1 First Attachment

Following tables show the transcription between IPA and ARPAbet for vowels (Figure A.1) and consonants (Figure A.2). Note, that IPA diphthongs are not transcribed separately but as one two-character ARPAbet symbol.

ARPAbet	IPA	Example
AA	ɑ	balm, bot
AE	æ	bat
AH	ʌ	butt
AO	ɔ	story
AW	aʊ	bout
AX	ə	comma
AXR	ər	letter
AY	aɪ	bite
EH	ɛ/e	bet
ER	ɜr	bird
EY	eɪ	bait
IH	ɪ	bit
IX	ɨ/---	roses, rabbit
IY	i	beat
OW	oʊ	boat
OY	ɔɪ	boy
UH	ʊ	book
UW	u	boot
UX	ʉ/---	dude

Table A.1: Vowel phonemes - transcription between IPA and ARPAbet.

ARPAbet	IPA	Example

Table A.2: Consonant phonemes - transcription between IPA and ARPAbet.