Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Pfaffenwaldring 5B
D-70569 Stuttgart

A Rapper's tool

Grigorij Aronov
Master Thesis

Prüfer: Dr. Antje Schweitzer
Prof. Dr. Grzegorz Dogil

Betreuerin: Dr. Antje Schweitzer

Beginn: 12.04.2017
Ende: 11.10.2017

Statement of Authorship

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe. Alle Zitate und sinngemäßen Entlehnungen sind als solche unter genauer Angabe der Quelle gekennzeichnet. Die eingereichte Arbeit ist weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen. Sie ist weder vollständig noch in Teilen bereits veröffentlicht. Die beigefügte elektronische Version stimmt mit dem Druckexemplar überein.

Grigorij Aronov

Contents

1 Abstract

The aim of this thesis is to examine different means for extracting rhymes to be used by lyricists or poets. While regular rhyme dictionaries usually use rhyme lexicons, here as data source any written text can be used (e.g. prosa, newspaper articles,...). The method presented here does *not* require an annotated corpus or a corpus explicitly containing rhymes. The main focus lies in finding a good metric to extract rhymes. This is done using phonological features to compare sounds. Although the thesis was written with German rap lyrics in mind, it can be useful to any music or non-music genre where rhymes are used. Furthermore, it is easily modifiable to be used with any other language, as the parameters that have to be changed are mostly based on phonetic features of the IPA.

2 Introduction

People engaged in writing rap lyrics constantly try to find new rhymes which nobody has ever used before. For classical poets there are rhyme dictionaries which they can use as a helping device to write poems. They may even be used to write all kinds of lyrics for different music genres. However, for writing rap lyrics such dictionaries are inapt. The author of the thesis does not have any knowledge of dictionaries *not* created by people, manually entering rhymes into a corpus. Here, by design, no *new* rhymes can be created because someone already invented them and added them to the database. Another issue emerges when considering the different types of rhymes (discussed in 4), especially *slant rhymes*. Slant rhymes are rhymes that sound similar but not entirely perfect. Considering that these kinds of rhymes have established themselves as a standard in lyrics, the thesis attempts to answer some of the following questions:

- What can be accepted as a rhyming syllable?

As slant rhymes are imperfect rhymes, how imperfect can a rhyme get before it is not considered a rhyme anymore?

- How can rhyme quality be measured?

Why does **car** rhyme with **core** and does it rhyme worse than with **bar**?

- How is the production of slant rhymes connected to its transcription? (e.g. "word bending")

The performer could say something like **ne-herd** instead of **nerd** for example to rhyme it with **forward**.

- How does the number, order, position and type of consonants in a syllable change the rhyme quality?

If **rank**/**rant** rhymes, how much quality is lost by replacing [k] with [t]?

After answering those questions (to a more or less satisfactory grade) a new approach can be made to find rhymes, while not requiring a corpus with annotated rhymes or rhymes that were placed into that corpus explicitly. In other words, rhymes can be extracted from any non-rhyming corpora, such as fiction books or manuals for toasters. A proof-of-concept script was written to show that rhymes can indeed be extracted from such corpora:
A grapheme-to-phoneme converter is used to transcribe the corpus into a phonetic representation. Then, an input string can also be converted and broken down into its components (such as syllables, nuclei, codas, vowels and consonants) and compared to strings in the corpus. The main purpose of this thesis is to examine metrics that can be used to estimate the similarities between these components, resulting in the ability to find out if a string is a (good) rhyme. An important part in identifying a usable similarity metric for

rhymes is to analyze the basic elements in speech. While for vowels it was clear how similarity could be assessed, for consonants a perception experiment was conducted.
The following sections are introducing phonological basics and explain certain kinds of rhymes. Then an overview of the related work is given and different similarity metrics are presented for different kinds of sounds. Subsequently, the implementation is explained and the results of the implementation are discussed. The outlook shows possibilities to improve the algorithm and to expand its functionality to not just stay *A Rapper's Tool* but to become *A Rapper's Toolkit*.

3 Phonological basis

The thesis analyzes several metrics to estimate how well two sequences of syllables rhyme by analyzing the similarities of sounds. This section gives a short overview over phonological basics that are required to understand how the similarity between syllables was assessed in this thesis.

3.1 Syllables

A syllable can be modeled as a hierarchical tree structure (see Figure 1). In that tree, at the first level there is the *onset* and the *rhyme*. The onset consists of consonants appearing on the left of the vowel of the syllable, while the rhyme consists of the *nucleus* and the *coda*. The nucleus is usually a vowel or a diphthong and the coda consists of all consonants that appear after the nucleus. The onset as well as the coda can also be empty.

3.2 Vowels

Figure 2 shows a vowel diagram of the German language. Vowels are commonly modeled by three dimensions, namely height or closeness (vertical

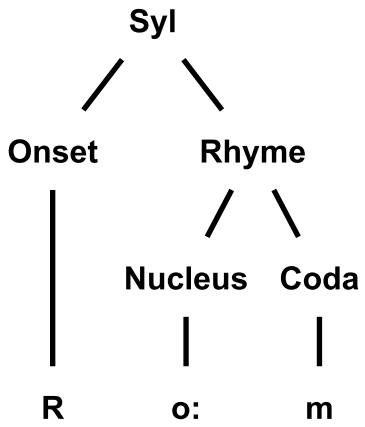


Figure 1: Hierarchical model of a syllable

axis), backness (horizontal axis) and roundness (for most vowels there are unrounded ([i]) and rounded ([y]) versions).

3.3 Consonants

Consonants commonly are measured in three dimensions, namely place, manner and voicing. Table 1 shows all German consonants grouped by these three dimensions. The rows denote the manner in which the consonants are produced, while the columns denote the place they are produced in. For most places of production a phoneme can be produced in an unvoiced (uv.) and in a voiced (v.) manner.

3.4 Syllable weight

A different way to represent syllable structure is offered by moraic theory. Mora [μ] is a unit used to determine syllable weight. A light syllable usually has one mora, while a heavy syllable has two. A syllable usually cannot

	bilabial		labio-dental		alveolar		postalveolar		palatal		velar		uvular		laryngal	
	uv.	**v.**	**uv.**	**v.**	**uv.**	**v.**	**uv.**	**v.**	**uv.**	**v.**	**uv.**	**v.**	**uv.**	**v.**	**uv.**	**v.**
Plosive	p	b	-	-	t	d	-	-	-	-	k	g	-	-	ʔ	-
Fricative	-	-	f	v	s	z	ʃ	-	ç	-	-	-	χ	ʁ	h	-
Affricate	-	-	p͡f	-	t͡s	-	t͡ʃ	-	-	-	-	-	-	-	-	-
Approximant	-	-	-	-	l	-	-	-	j	-	-	-	-	-	-	-
Nasal	m	-	-	-	n	-	-	-	-	-	ŋ	-	-	-	-	-

Table 1: German consonants in IPA

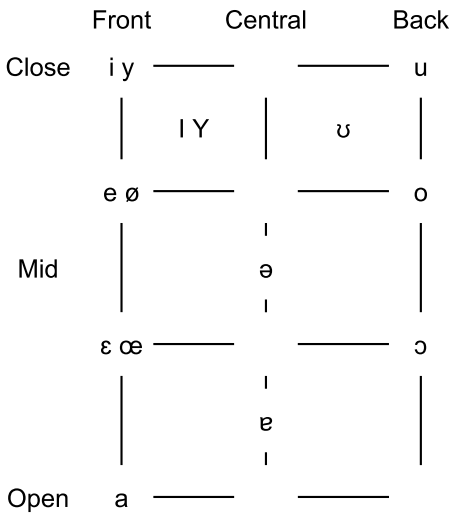


Figure 2: German vowel diagram

have more than two morae (Hall, 2011). A short vowel is associated with one mora, while a long vowel is associated with two. The coda is usually assigned a mora, while the onset never is assigned one. The possible syllable combinations that can occur are the following: CV, CV:, CVC and CV:C, as can be seen in Figure 3. In languages like Latin or English only CV is considered to be a light syllable (1μ), while all others are considered heavy syllables (2μ). In some languages, like Mongolian, CVC can also be a light syllable. For diphthongs, each half is connected to its own mora.

4 Rhymes - an overview

Poetry has been there for thousands of years. One of the earliest known poetry books is the Shijing, which is dated roughly between the eleventh and the seventh centuries B.C. (Baxter, 1992). This is also one of the first works in which rhymes appear. For at least three thousand years rhymes were used as an important stylistic device in poetry to convey meaning, emphasize

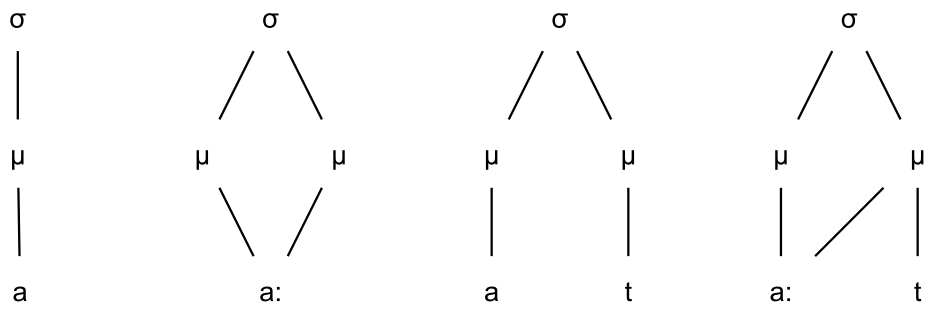


Figure 3: Possible syllable structures, according to Hall (2011)

important parts of a poem or just for the enjoyment of saying or hearing rhymes. However, despite their importance, rhymes have not evolved very much until recently and stayed rather "simple".

4.1 Rhyme

Rhymes are hard to define, as there are so many different kinds of rhymes. According to Simpson and Weiner (2016) a rhyme is the correspondence of sound between words or their endings. However, it is not very clear what "correspondence" means here. It could either mean that the words or their endings must sound similar or it could mean that the words or their endings must be identical. While in **flower** and **power** the ending of the words is identical and therefore they rhyme, it does not appear sensible to say that **flower** rhymes with **flower** due to their "correspondence". The word pair **flower**/**flower** may correspond to all definitions of a rhyme, however the mere repetition of a word is not desirable when searching for rhymes. It is also unclear what "similar" would mean, as it would need a metric to measure similarity of syllables (are **fine** and **lines** similar? If they are, do they rhyme?). In Farlex (2016) one of the definitions is that rhymes are corresponding terminal sounds of words or of lines of verse, meaning that the rhyme in **flower** and **power** is only the second syllable **-wer**, although **flo-** and **po-**

are rhymes too. Reddy and Knight (2011) define a rhyme as two words with *identical* final stressed vowels followed by identical phonemes. This corresponds to the definition in Farlex (2016) but also to the general definition in phonology, if "corresponding" is defined as being identical. Phonology defines a rhyme as the part of a syllable, consisting of a nucleus (usually a vowel or diphthong) and a coda (all consonants after the nucleus). In Figure 1 the nucleus is the /oː/ and the coda is the /m/. Together they constitute the phonological rhyme. In other words, two syllables rhyme when their rhymes are identical.

The Japanese folk definition of a rhyme only considers matching vowels and explicitly says that consonants play no role in rhymes, according to Kawahara (2007). However, the same paper finds evidence that consonants do play a role.
Such definitions cannot represent adequately what rhymes are, as they often are imprecise, tend to contradict each other and are not applicable to different kinds of rhymes, such as similes or multisyllabic rhymes (explained in the following sections). In this thesis the word "correspondence" is used only to refer to the *position* of a syllable and not to its rhyming nature. The last syllable of verse *a* corresponds to the last syllable of verse *b* regardless of whether these syllables rhyme.

In the following sections different types of rhymes are presented. The definitions of the rhyme types may overlap, however it is important to note that a definition only refers to certain but not to all properties of a rhyme.

4.2 Single Rhymes

One example of what is meant by a "simple" rhyme (as mentioned above) can be seen in the following examples:

William Shakespeare, Sonnet 22

Presume not on thy heart when **mine** is **slain**;
Thou gavest me **thine**, not to give back a**gain**.

Edgar Allan Poe, A dream within a dream

O God! can I not **grasp**
Them with a tighter **clasp**?

These rhymes are called *masculine rhymes* or *single rhymes*, as they are monosyllabic, i.e. only one syllable of a word rhymes with only one syllable of another word (**grasp - clasp**). For the lack of a well-established definition, rhymes will be called *simple* when they are mono- or disyllabic (for an example see 4.5). In rap sometimes simple rhymes are condescendingly referred to as *weak rhymes*.

4.3 Perfect rhymes

In a *perfect rhyme* the stressed vowel sound and all subsequent sounds in two words are identical and the sounds preceding the vowel must be different (**skylight/highlight**) (Davis, 1985).

4.4 Slant rhymes

Slant rhymes, also known as *half rhymes*, *inexact rhymes*, *oblique rhymes*, *approximate rhymes*, *near-rhymes*, *lazy rhymes*, *imperfect rhymes* and many more, are rhymes with similar but not identical sounds (Davis, 1985).

The Hives, Dead Quote Olympics

This time you really got something, it's such a clever **idea**
But it doesn't mean it's good because you found it at the **liba-ra-ri-a**

George Wolff, To My Wife

If love is like a **bridge**
or maybe like a **grudge**,

4.5 Double Rhymes

Double rhymes, *feminine rhymes* or *disyllabic rhymes* are sequences of two or more syllables where each syllable of one syllable sequence rhymes with each corresponding syllable of another syllable sequence.

William Shakespeare, Sonnet 20

But since she prick'd thee out for women's **pleasure**,
Mine be thy love and thy love's use their **treasure**.

Here, **plea-** rhymes with **trea-** and **-sure** in both words is identical and therefore is also considered a rhyme. Note that, while in 4.1 the repetition of a word is not considered a rhyme, this does not apply to the repetition of a syllable.

Double rhymes sometimes are used to describe the following rhyme types.

4.6 Triple rhymes

Triple rhymes or *trisyllabic rhymes* are rhymes with three subsequent syllables rhyming with three subsequent syllables of another syllable sequence (**fearfully/tearfully**).

4.7 Composite rhymes

Mosaic rhymes or *composite rhymes* are rhymes consisting of more than one word (**interstellar/in the cellar**).

4.8 Multisyllabic rhymes

Multisyllabic rhymes describe all kinds of rhymes with more than one rhyming syllable, implicating double, triple, quadruple,... and mosaic rhymes in all their qualities (perfect, slant and others).

Gerard Manley Hopkins, The Bugler's First Communion (1918)

This very very day came down to us after **a boon he on**

...

Came, I say, this day to it—to a First **Communion**.

In **a boon he on//Communion** the rhyme is a multisyllabic rhyme with four syllables. This means that each syllable of the first syllable sequence rhymes with a corresponding syllable of the second syllable sequence.

/ə/	/buːn/	/hi/	/ɔn/
/kəm/	/juː/	/ni/	/jən/

The last syllables (/ɔn/-/jən/) seem not to rhyme very well as they sound too different. A solution would be to pronounce one of the syllables in a very strange way. One could either pronounce the [ɔ] in [ɔn] as an [ə] (/ɔn/ → /ən/) or as something that is more similar to an [ə], such as [e] (/ɔn/ → /en/). Or one could change the [ə] in [jən] to an [ɔ] (/jən/→/jɔn/) or something similar to it, such as [o] (/jən/ → /jon/). However, this is not an issue, as will be explained in the next section.

Although multisyllabic rhymes (with a small number of syllables) were used by poets for quite a long time, these kinds of rhymes have never gotten much attention and they are an uncommon stylistic device in classical poetry.

In the 1970s Hip hop culture started to evolve and with it rap, as a part of the culture (Chang, 2007). At so called "Block Parties" DJs started reciting their rhymes over beats of pop songs. Much of the rhymes were improvised and therefore simple. In the course of time this evolved into rap. For the simplicity of the lyrics this period is called *old-school rap*. In the 1980s and 1990s rap drastically evolved regarding its style, and rap artists started to use multisyllabic rhymes. By 1992 Kool G Rap used rhymes with up to six syllables:

Kool G Rap and DJ Polo, Letters

I pop bad cops, I got **a pig a day habit**
Bing bing bang, just like **the ricochet rabbit**

The *golden age* of rap ended in the mid 1990s.

Today multisyllabic rhymes, or *multies*, as they are called in the rap scene, are a standard and a means to show a rapper's skills. The number of syllables in a rhyme is open-ended and they are used in combination with many other literary techniques which are commonly found in classical poetry, such as homophones, similes, alliterations and metaphors.

Since multisyllabic rhymes are in the center of this work, all further considerations are done looking at rhymes as *rhyming syllable sequences*.

Long single word rhymes, especially long noun rhymes are considered to be harder to find than sequences of short words. The German language is special in this regard as it allows compounds which are quite long:

Kollegah, V.I.P.I.M.P.

Geht mir **Kaltgetränke bringen** und dann **Palmenblätter schwingen**
Ich brauch 'nen kühlen Kopf beim Business mit Ge**waltverbrecherringen**

This makes German an interesting language to look at for multies.

4.9 Word bending

An often used stylistic device in rap is sometimes referred to as "*word bending*", which for example Eminem makes use of excessively:

Eminem, Lose Yourself

It only grows **harder**, homie grows **hotter**
He blows. It's all **over**. These hoes is all **on him**

While in these two lines every syllable is a rhyme and all the rhymes have five to six syllables, some words are usually not considered proper rhymes.

However, they are "bent" or produced in a manner that makes them more similar, by changing some phones:

harder	/haɹdəɹ/	→	[haɹda]
hotter	/hɒtəɹ/	→	[hɒɾa]

In this case [t] was flapped in **hotter** to make it more similar to [d], which also occurs in **harder**.

over	/oʊvəɹ/		[oʊvə]
on him	/ɑn əm/	→	[ɔan əm]

In **on him** he also seems to slightly prolong the **on**, almost making it an [ɔa] diphthong to match the diphthong in **over**. It may also be noted that Eminem is from northern USA, where [ɑn] is rather used than [ɔn] which is used in Southern American accent. Eminem however uses the southern American version in this case, making the syllable more similar to the [o] in **over**.

As this example shows, dialects may be helpful too. E.g. in some areas in the US speakers tend to pronounce [ɑ] in a rather closed manner, so that **car** may rhyme with **core**.

While this is by far not everything that can be said about rhymes, this is all that is needed for the thesis with regard to rhyme types.

5 Related Work

There are two general approaches to find rhymes in a corpus. By training probabilistic models using a corpus with annotated rhymes, in order to learn to predict rhymes in a test set (Reddy and Knight, 2011) (described below in more detail) or by estimating the similarity of phonemes using phonological features. The similarity is then used to compute how well two syllables rhyme. Some works using one of these two approaches are presented in this section.

Kawahara (2007) analyzes *half rhymes* or *slant rhymes* (rhymes with non-identical but similar sounds) in a Japanese rap lyrics corpus and provides evidence that the *similarity* of consonants in the coda is strongly connected to the likelihood of making a rhyme pair (*rhymability*). This conflicts with some classical definitions of a rhyme, defining it as two words with *identical* final stressed vowels followed by *identical* phonemes (Reddy and Knight, 2011).
Kawahara (2007) supports the idea that speakers use knowledge about psychoacoustic similarity[1] with properties that are not covered by featural similarity, thus suggesting that psychoacoustic similarity may be more relevant. Furthermore, he introduces a way to measure *rhymability*. He also provides a way to measure the salience of phonetic features, to indicate which features are more important than others. However, due to feature overlaps only some significant differences in salience of the features could be extracted. [palatalized] appeared to be very salient, [voiced] appeared to be less salient and [nasal], [continuant], [sonorant], [consonantal] and [place] turned out to have an effect that is too weak to be detected (using multiple regression). Further approaches to measure salience of features were made by Kessler (1995) who assigned arbitrary values between 0 and 1 to phonetic features and by Nerbonne and Heeringa (1997) who used binary features.

In McCurdy et al. (2015) a formalism to explore sonic devices in poems is provided. Among other things it allows to search for rhyme patterns in poetry. For example a perfect masculine rhyme **O[NC]'** has an onset **O**, a nucleus **N** and a coda **C**. **NC** is the rhyming segment (denoted by the brackets) and the rhyming segment is required (denoted by the apostrophe). As a proof of concept they present an application called "RhymeDesign". The tool searches for such patterns in a poem.

A lot of research exists providing methods to find all kinds of rhymes in poems or lyrics (Reddy and Knight, 2011; Hirjee and Brown, 2009; 2010;

[1]Psychoacoustic similarity is similarity that is perceived based on detailed acoustic information, as opposed to simple features (Kawahara and Shinohara, 2009).

Addanki and Wu, 2013). Reddy and Knight (2011) propose an unsupervised approach without knowledge about rhyme or pronunciation to discover rhyme schemes by learning rhymes from a rhyming corpus and finding these rhymes in stanzas. This approach assumes rhyme sparseness and that rhymes are often used between poets. This contradicts the assumptions in this thesis, as the extensive use of slant rhymes mitigates the rhyme sparseness problem severely. It would be hard to train slant (multisyllabic) rhymes, as such corpora for German are rare and the quality or "slantness" of a rhyme may be very subjective. Furthermore, the paper does not avoid nonsense rhymes, which this thesis explicitly tries to avoid.

Hirjee and Brown (2010) proposed a method to score potential rhymes based on a probabilistic model counting phoneme frequencies. The task of finding (multisyllabic) rhymes in lyrics however, is different to finding (multisyllabic) rhymes in general, to imitate a rhyme dictionary. Lyrics often have a rhyme structure that is more complex than "some words followed by a multisyllabic rhyme at the end of the verse". The rhymes may be scattered over several verses in different places in the verses and alter with other rhymes.

Eminem, The way I am

Getting this stress that's been eating me recently off of this chest
And I rest again peacefully

Here, Eminem rhymes **Getting this**/**stress that's been**, which is in context with his flow (three syllables at a time) not a multisyllabic rhyme but a *rhyme chain* (two rhyming syllable sequences without intermediate words). Then he changes to **eating me**/**recently**, changing to **of this chest**/**And I rest** and then going back to rhyming **stress that's been eating me** (for this the rhyme chain indeed can be considered a multisyllabic rhyme) from the beginning with **rest again peacefully**. Meanwhile, **chest** is also part of the rhyme (**Get**ting/**stress**). Furthermore, the verses are full of vowels that Eminem plays with, going from [ɛ], to [i], to [ə] and back to [i] again, which, according to him is often his approach to write lyrics in the first place.

Here, you can even see some ambiguity with regard to what the syllable sequences rhyme with. Searching for rhymes in such lyrics is difficult and may work to a degree for Hirjee and Brown (2010). However, it is not the aim of the thesis to be able to find such complex rhyme structures in a text and it is unacceptable for a rhyme dictionary to yield wrong results that are possible in a probabilistic approach. The model must guarantee that a rhyme is indeed a rhyme. Here, this only can be achieved by manually defining a threshold that distinguishes whether something is or is not a rhyme.

Within several approaches to define a metric for phoneme similarity, Bailey and Hahn (2005) found the simple counting of features to be the best way to predict similarity. They found that differences in manner have more influence on phoneme similarity than voicing or place of articulation.

6 Possible Metrics

This section looks into different ways to quantify rhyme quality in a way that is applicable to the goal set in the thesis and that is applicable to most languages. One way to measure the quality of a rhyme is examining all its components - phonemes in this case. In theory more precise results would be possible using phones. Phones are realizations of phonemes that are not critical for the meaning of a word, while different phonemes do change the meaning. However, the data used in this thesis is written language which does not allow for such a distinction. Instead, the phonemes can easily be looked up in a lexicon. In literature the most popular (general) approach to measure phoneme similarity seems to be based on phonological features. There, similarity is defined as the difference in features.

Another possible approach is to acquire the needed information from statistical analysis of phoneme similarity in experiments where the participants compare words or sounds, resulting in confusion matrices for phonemes.
While there appears to be a good number of confusion matrices for phonemes

extracted from different contexts for English, it seems to be hard to find them for German. The experiment described in 6.7 attempts to order consonants by their similarity. In theory, this could be used to build a confusion matrix for the consonants, however the experiment is not representative enough to allow to infer similarity scores from the results.

Probabilistic models do not require a lot of previous knowledge about rhymes. What is needed however, is a corpus that contains well annotated rhymes. These seem to exist but they are hard to find for the German language and usually do not distinguish between slant rhymes and perfect rhymes. Therefore, it is difficult to score the rhymes that are extracted by quality. The probability with which the extracted string is a rhyme can be taken as a way to score the results but such models usually also yield wrong results with a certain probability. As the main task in the thesis is to imitate a rhyme dictionary for multisyllabic rhymes, "wrong" results can not be allowed. It is also unclear how a test set would look like, as it would have to be a manually annotated corpus of (prosaic) books with rhyming syllable sequences. This definitely does not exist and it is unrealistic to create such a gold standard.

These are the reasons why in this thesis features are used to extract rhymes.

Looking at all definitions of a rhyme (in poetry) mentioned in 4.1, the lowest common denominator seems to be that a syllable rhymes when the nucleus and the coda in two syllables are similar. In the following general metrics for syllable similarity are constructed based on this weak definition.

6.1 Similarity of Onsets

The onset is not considered in this thesis, as no known definition considers it to be important for a rhyme. It does not seem to matter which or how many consonants are in the onset, the rhyme still appears to be equally good (**and/land/strand**).

6.2 Similarity of Vowels

The vowel is the sound carrying the most information to identify a word (Menon, 2006) and therefore it is the most important sound in a rhyme. Simply changing the vowel in a word (**list**/**lost**) changes the sound in a greater way compared to changing a consonant (**list**/**lisp**). The heuristics in the thesis however, do not simply discard a rhyme when the vowels are not identical in two syllables. Two approaches to measure vowel similarity were examined.

Approach 1 Figure 4 shows the vowel diagram of the German language (introduced in section 3.2) with the common three dimensions. Intuitively, slightly changing only one of these dimensions should not make a huge difference in the sounds. Therefore, the metric for similarity of vowels based on this diagram can be defined as follows:

$$D_{n_1,n_2} = \begin{cases} 0 & \text{if } d_{n_1,n_2} = 0 \\ 1 & \text{if } d_{n_1,n_2} = 1 \\ \infty & \text{if } d_{n_1,n_2} > 1 \end{cases} \tag{1}$$

Where D_{n_i,n_j} is the **D**istance defined by the metric, n_1, n_2 are the nodes in the vowel diagram and d_{n_i,n_j} is the number of edges between the nodes n_i, n_j. In other words, if the vowels in two syllables are the same ([ɛ]/[ɛ]), the distance is 0, if the vowels are neighbors in the graph ([e]/[ɛ]), then the distance is 1 and if the nodes are further apart than one edge ([e]/[a]), then the syllable is discarded as a possible rhyme. However, while between close and open and between front and back there are intermediate steps, those are missing for the rounding dimension. Thus, the difference between rounded and unrounded may be too big to consider them similar.

Approach 2 The positions of the vowels in the vowel diagram are derived from the frequencies of their formants. Thus, $F1$ corresponds to openness

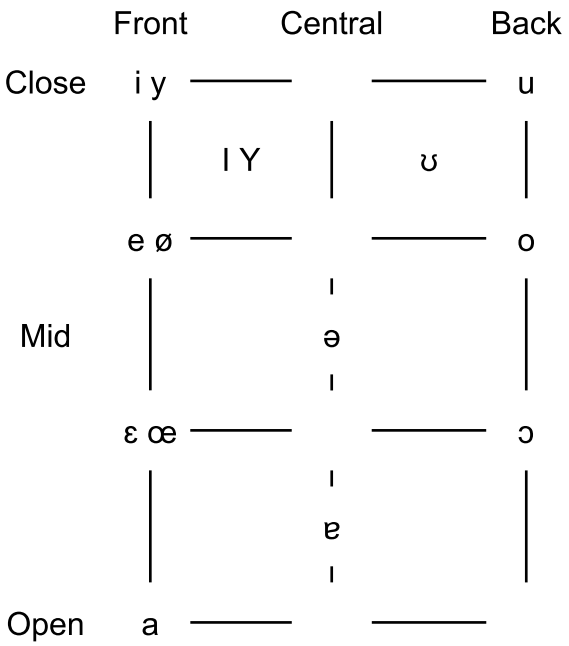


Figure 4: German vowel diagram

while $F2$ corresponds to backness (Cairns et al., 2010). This fact can be easily exploited to measure similarity even more accurately (in theory). A simple approach is to take existing formant measurements for speakers of a certain language, to normalize them and to then calculate the Euclidean distance between the formants of two different vowels. For this thesis, the median values for formants of male German speakers were taken from Pätzold and Simpson (1997). To accommodate human perception (see Table 3) the formants were converted to the mel-scale. This also solves the problem of roundness being a binary feature, as roundness is reflected in $F2$ and higher formants (Padgett, 2001). $F3$ also plays a role, making the estimation more precise as it contains information about the backness of a vowel (Johnson, 1989).

However, formants are able to influence each other (Padgett, 2001) so that it becomes hard to quantify the dimensions reasonably. To reduce the imbalance caused by formants which are less salient, only the first three formants were considered when calculating the distance between vowels. Assuming that the third formant is less important, its distance between two mel-frequencies was divided by two. The distance between the mel-frequencies of the first three formants of two vowels was calculated as

$$D_{mel} = \sqrt{(F1_1 - F1_2)^2 + (F2_1 - F2_2)^2 + \left(\frac{F3_1 - F3_2}{2}\right)^2}.$$

The distances between mel-frequencies for all vowels are shown in Table 2. A threshold to distinguish between similar and non-similar vowels was applied at $D = 200$, after examining results for tens of different random words. The results appeared to be too dissimilar when D was too high, while well sounding results disappeared when D was too low.

To keep the scores consistent with the scoring for consonants and with the scoring in **Approach 1**, the mel-frequency distances were mapped to the range of scores that were obtainable in **Approach 1**. The smallest value possible was 0 when the vowels in a vowel pair were the same, the biggest

	iː	ɪ	yː	ʏ	eː	ɛ	øː	œ	a	aː	oː	ɔ	uː	ʊ	ɐ	ə
iː	0	96	276	311	111	297	306	291	599	540	688	587	615	545	395	255
ɪ	**96**	0	215	234	58	207	224	199	503	444	611	498	548	465	299	170
yː	276	216	0	73	266	277	97	175	429	389	419	349	343	281	246	98
ʏ	311	234	**73**	0	275	234	34	124	356	317	382	287	322	236	176	71
eː	**111**	**58**	266	275	0	192	260	215	509	447	650	525	593	503	311	208
ɤ	297	207	277	234	**192**	0	205	114	327	262	549	388	523	405	153	184
øː	306	224	**97**	**34**	260	206	0	93	338	296	397	288	345	249	152	63
œ	291	**199**	**175**	**124**	215	**114**	**92**	0	311	256	458	317	421	310	109	91
a	599	503	429	356	509	327	338	311	0	65	381	196	425	299	206	372
aː	540	444	389	316	447	262	295	256	**65**	0	407	217	436	304	152	323
oː	688	611	419	382	650	549	397	458	381	407	0	190	111	149	403	442
ɔ	587	498	349	287	525	388	288	317	**196**	217	**190**	0	231	116	235	333
uː	615	548	343	322	593	523	345	421	425	436	**111**	231	0	133	391	386
ʊ	545	465	281	236	503	405	249	310	299	304	**149**	**116**	**133**	0	264	296
ɐ	395	299	246	**176**	311	**153**	**152**	**109**	206	**152**	403	235	391	264	0	172
ə	255	**170**	**97**	**71**	209	**184**	**63**	**91**	372	323	442	333	386	296	**172**	0

Table 2: Euclidean distances between the mel-frequencies. Bold values are $<$ 200. The matrix is symmetrical.

value was 1. Additionally, the score can increase by up to 0.5 when considering morae and by another unit of 1 if considering diphthongs (next section). Therefore, the mel-distances between 0 and 200 were mapped to a range between 0 and 2.5 by the formula

$$f(x) = \frac{(a-b) * (x - min)}{max - min} + a$$

where $[a, b]$ is the score range that the mel-distances are mapped to (here $[0, 2.5]$), $[min, max]$ is the range of values that are mapped to $[a, b]$ (here $[0, 200]$) and x is the mel-distance between two vowels that is being mapped to a value between a and b.

6.3 Similarity of Diphthongs

Diphthongs, such as [aʊ], [aɪ], [ɔʏ] and [ʊɪ] are treated separately. Their neighborhood distances are distinguished by looking at the neighborhood of the first and the second vowel. Similar diphthong pairs are selected by comparing their first vowel.

male	F1	F2	F3
i	434	1646	1813
ɪ	500	1577	1835
y	448	1373	1742
ʏ	516	1347	1758
e	541	1618	1836
ɛ	691	1498	1835
ø	544	1363	1738
œ	616	1420	1761
a	843	1209	1809
o	548	968	1825
ɔ	674	1110	1822
u	457	1031	1801
ʊ	559	1116	1797
ɐ	689	1345	1823
ə	530	1410	1817

Table 3: Mel-frequencies for male F1, F2 and F3. Original Frequencies taken from Pätzold and Simpson (1997)

The diphthong pair [aʊ]/[aɪ], has a distance of 1, as both diphthongs have one identical vowel pair ([a]/[a]), while the non-identical vowel pair is not in its mutual neighborhood ([ʊ]/[ɪ]). Pairs such as [aʊ][ɔɪ] would not be considered a rhyming pair, as [a]/[ɔ] are not identical or in their mutual neighborhood. This can be formalized according to Equation 2,

$$DD_{(n_1,n_2),(n_3,n_4)} = \begin{cases} 0 & \text{if} \quad D_{n_1,n_3} = 0 \quad \wedge \quad D_{n_2,n_4} = 0 \\ 1 & \text{if} \quad \begin{cases} D_{n_1,n_3} = 0 \quad \wedge \quad D_{n_2,n_4} > 0 \\ D_{n_1,n_3} > 0 \quad \wedge \quad D_{n_2,n_4} = 0 \end{cases} \quad \vee \\ \infty & \text{if} \quad D_{n_1,n_3} > 0 \quad \wedge \quad D_{n_2,n_4} > 0 \end{cases} \tag{2}$$

where DD is the **D**iphthong **D**istance, (n_1, n_2) and (n_3, n_4) are diphthongs with n_1 and n_3 being the first vowels in the diphthongs and n_2 and n_4 being the second vowels. D_{n_x,n_y} is the distance between vowels which were defined in Equation 1. For syllable pairs where one syllable has a diphthong and the

other syllable does not, the distance is 1 if one of the vowels in the diphthong is in the neighborhood of (or identical to) the vowel in the syllable without a diphthong.

The easiest way to adapt diphthongs to the second approach for vowels would be to default to the formants of the first vowel. However, it turned out to have a very negative impact on rhyme quality, which is why the second vowels in a diphthong pair also needed to be considered. As its importance is smaller, the mel-distance between the second vowels of the diphthongs was halved and added to the distance of the first vowels in the diphthong pair. For example the distance between [au] and [aɪ] is calculated as $0+548/2 = 274$, as the mel-distance between the [a]s is 0, the distance between the [u] and the [ɪ] is 548, which has to be divided by 2 because the vowels are at the second position. Likewise, if the first nucleus is a diphthong and the second nucleus is a regular vowel, the distance is calculated as the sum of the distance between the regular vowel and the *first* vowel in the diphthong and the distance between the regular vowel and the *second* vowel in the diphthong. For example [e] and [ea] would have the distance $0 + 509/2 = 254.5$, as the distance between [e] and [e] is 0, the distance between [e] and [a] is 509, which has to be divided by 2, since [a] is at the second position in the diphthong.

6.4 Metrics for syllable weight

Moraic Theory can be used as another feature to rate similarity between syllables. In the case of German this would influence the similarity score as follows: If two syllables both end with a short vowel or if two syllables both do not end with a short vowel, then they are similar. Else the similarity decreases. Or, according to moraic theory: if for both syllables $\mu = 1$ or if for both syllables $\mu = 2$ then they are similar. Else the similarity decreases. Another way would be to only consider how many morae are connected to the vowel. In this case for Figure 3 it would mean that the second and the fourth tree would be connected to two morae, while the first and the third one

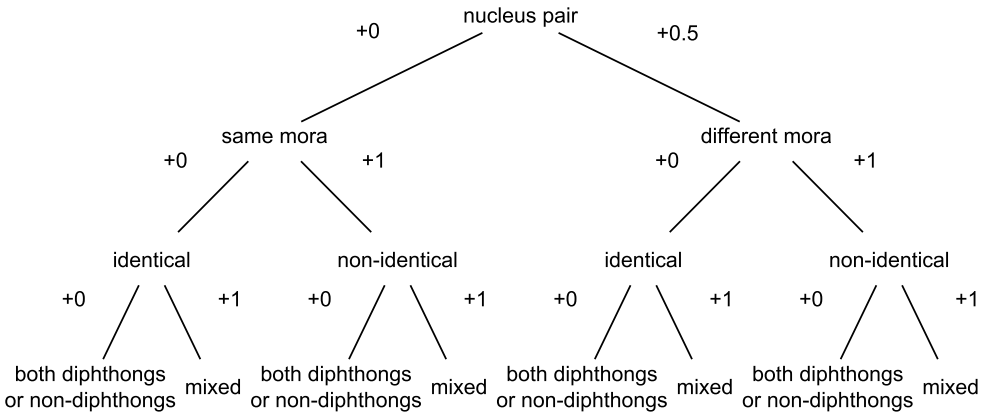


Figure 5: Calculating the score for a nucleus pair

would only be connected to one. Essentially, this would mean that the coda is irrelevant and only vowel length is important. While both approaches deal with vowel length (which was not yet discussed in the thesis), it is neither clear which of these approaches is more reasonable nor how high the penalty for the similarity score should be. Considering all phenomenons described until now, like word bending or flow, both being able to change the sound of a word in a manner that does not hurt rhyme quality, a discrepancy in vowel length does not seem to make a huge difference. A penalty of 0.5 seems appropriate.

6.5 Summary for vowels

Calculating the vowel scores for **Approach 1**, considering diphthongs and morae can be represented as a decision tree, as can be seen in Figure 5.

In the leftmost path a nucleus pair is assigned an overall score of 0 if it has the same moras, identical vowels and if both nuclei are or are not diphthongs. In the rightmost path a nucleus pair is assigned an overall score of 2.5 if the nuclei have different morae, non-identical vowels and if one of the nuclei is a diphthong and the other is not.

For **Approach 2** morae and identity for the nuclei is implied. Therefore, just the mel-distance between the nuclei and the knowledge whether either of the nuclei is a diphthong is needed. If one of the nuclei is a diphthong, the corresponding mel-distance is added to the vowels in the first position.

6.6 Similarity of Consonants

There are several approaches to assess similarity in consonants:

In line with the method of measuring similarity of vowels, the similarity of two consonants can be measured by examining its dimensions. One possibility is to look at the distances between the places of articulation in the vocal tract (Croft, 2002). If two consonants are produced in the same place, ([p]/[b]) then the consonants are similar. If the places of articulation are far apart, ([p]/[ʁ]) they are dissimilar. However, this method only considers the place of articulation, assuming that manner is not important for similarity. Furthermore, such a hierarchy can only be established for three of the most common palaces of articulation, according to Croft (2002). The hierarchy would be

$$bilabial > dental/alveolar > velar.$$

Another approach is to look at manner by way of the "consonanthood" of the consonants. According to Croft (2002); Berg (1998) consonants can be sorted by their consonanthood or their sonority. In other words consonant similarity is measured by analyzing how similar a consonant is to a vowel. According to this theory consonants can be arranged as follows:

$$glides > rhotic > lateral > nasals > voiced\,fricatives$$

$$> voiceless\,fricatives > voiced\,stops > voiceless\,stops$$

meaning that glides are more sonorant than rhotics and so on.

This in turn disregards the possible importance of the place of articulation as it only looks at articulation manner. In this thesis however, consonant similarity was measured by examining manner of articulation, as manner seems to have a higher influence on similarity, according to literature.

Another way to assess the similarity of consonants is to calculate a confusion matrix from complex psycholinguistic experiments, such as similarity judgment tasks under different conditions, for example noise/music etc. While in the following section a similar experiment was conducted, the experiment was rather simple and had too little participants to be used to acquire a good measurement for consonant similarity. Therefore, in the following only the theories around manner and place are compared.

6.7 Analyzing consonant similarity theories

To measure consonant similarity by their sonority or by their place of articulation, both seem to be sound approaches. To find out whether either of these approaches makes more sense, an experiment was conducted. In the following *Theory 1* denotes the idea that consonant similarity can be measured by analyzing their place of articulation while *Theory 2* denotes the idea that it can be measured by analyzing their sonority.

6.7.1 Setup

Stimuli with German consonants between the two vowels /a/ were recorded (/aba/, /apa/,...) and played in pairs to the participants. The participants had to judge the similarity of each of the pairs on a scale from 1 to 5 where 1 meant that the syllables are dissimilar and 5 meant that they are very similar. The order of the pairs played, as well as the order of the elements within each pair, were randomized. Each pair was played once to every participant, meaning that if, for example, /aba/ /apa/ was played, /apa/ /aba/ would not be played anymore.

The German language has 24 consonants. This means that $\frac{24!}{((24-2)!\cdot 2!)} = 276$ pairs of words would have to be rated by each participant. As this task is tedious and annoying, causing the participants to lose concentration quickly, it was decided to reduce the number of consonants. Out of 24 stimuli 14 were chosen for the experiment. That way each participant had to rate $\frac{14!}{((14-2)!\cdot 2!)} = 91$ pairs of stimuli. For all consonant classes that are present in German some consonants were kept, while consonants from classes where they are overrepresented (such as fricatives) or those which a non-linguist would not consider a consonant at all (such as [ʔ] or different kinds of /r/) were discarded. The discarded consonants were [ʔ], [d], [t], [h], [m], [ʃ], [ʁ], [x] and [ʒ]. While that reduced the annoyance for the participants, another problem was that some consonants never appear between two vowels in reality. The experiment however, tries to derive regularities, therefore syllables not appearing in reality may not be too problematic.

Experimental design. In a command line interface the 19 German speaking participants saw a short description of the task. They could read that in the experiment the similarity of consonants was analyzed and that they had to rate the similarity of syllable pairs. After pressing *enter*, the first two stimuli were played. Then, the participants were shown a new interface with the request to enter a number between 1 and 5, where 1 would mean not similar and 5 would mean very similar. After entering the number and pressing *enter* the next sound was played.

6.7.2 Results

Table 4 shows an ordered list of averaged participant ratings for all syllables. Higher average values mean that the syllables are considered more similar than syllables with lower average ratings. For example, it can be seen that [z] and [s] have an average rating of 4.78, which means that these consonants are considered very similar. It can be seen that for a small number of consonant

s1	s2	**avg**	s1	s2	**avg**	s1	s2	**avg**	s1	s2	**avg**	s1	s2	**avg**	s1	s2	**avg**	s1	s2	**avg**
z	s	**4.78**	l	f	**3.10**	f	b	**2.78**	v	ç	**2.68**	n	l	**2.52**	n	ç	**2.36**	z	j	**2.26**
ŋ	n	**4.42**	ŋ	g	**3.05**	v	g	**2.78**	l	g	**2.63**	n	f	**2.52**	ŋ	b	**2.36**	z	k	**2.26**
p	b	**3.73**	v	j	**3.05**	z	ŋ	**2.78**	ŋ	ç	**2.63**	ŋ	k	**2.52**	p	ç	**2.36**	ʁ	k	**2.21**
k	g	**3.63**	g	b	**3.00**	z	b	**2.78**	p	n	**2.63**	ʁ	ç	**2.52**	s	b	**2.36**	g	ç	**2.15**
v	b	**3.63**	l	b	**3.00**	j	g	**2.73**	ʁ	g	**2.63**	s	n	**2.52**	s	p	**2.36**	ʁ	j	**2.15**
v	f	**3.42**	s	j	**3.00**	l	ç	**2.73**	ʁ	f	**2.63**	v	ŋ	**2.52**	s	ʁ	**2.36**	s	ŋ	**2.15**
v	l	**3.42**	v	n	**3.00**	f	ç	**2.68**	z	n	**2.63**	v	p	**2.52**	l	k	**2.31**	ç	b	**2.10**
j	ç	**3.26**	z	f	**2.94**	j	b	**2.68**	ʁ	b	**2.57**	n	b	**2.47**	ŋ	f	**2.31**	k	j	**2.10**
p	k	**3.26**	n	j	**2.89**	j	f	**2.68**	v	s	**2.57**	ʁ	p	**2.47**	s	ç	**2.31**	s	l	**2.10**
p	f	**3.26**	ʁ	n	**2.89**	ŋ	j	**2.68**	v	ʁ	**2.57**	v	k	**2.47**	s	g	**2.31**	s	k	**2.10**
ʁ	l	**3.26**	g	f	**2.84**	ŋ	l	**2.68**	z	l	**2.57**	z	ʁ	**2.47**	n	g	**2.26**	z	p	**2.10**
s	f	**3.26**	z	g	**2.84**	p	g	**2.68**	k	b	**2.52**	z	ç	**2.42**	p	ŋ	**2.26**	n	k	**2.05**
l	j	**3.10**	z	v	**2.84**	p	l	**2.68**	k	f	**2.52**	k	ç	**2.36**	ʁ	ŋ	**2.26**	p	j	**2.05**

Table 4: Average ratings of all similarities between consonants

pairs at the beginning of the table the participants seemed to agree in their similarity ratings as they are much higher compared to the rest. Most of the ratings however are between 2 and 3, descending in a very gradual manner.

Theory 1. In Table 4 the first few highest rated pairs have the same place of articulation ([k][g], [s][z], [p][b]) or are produced in each others neighborhood ([b][v]). However, it can also be observed that [ŋ][ʁ] has an average rating of only 2.26. Although they are produced in each other's neighborhood ([ŋ] is velar, [ʁ] is uvular), they got rather low ratings. Furthermore, [l] and [ʁ] got a value of 3.26 although they are produced in higher distance to each other (alveolar vs. uvular).

In Figure 6 the ratings were averaged over all possible combinations of places of articulation and were sorted by the similarities that would be expected according to Theory 1. The vocal tract has two ends: on the one end there are the labiae and on the other end the glottis. Like in a double loop, starting from the labiae, the places of articulation are sorted by the distance to the location which is one step further apart in the direction of the glottis. Then the figure moves to the labiodental consonants and looks at the consonants which are produced next to it, moving towards the glottis again. Therefore, in Figure 6 one would expect that the bars descend between *bb* and *bv* (between pairs of bilabials and pairs of bilabials and velars) then jump to a higher rating at *ll* (pairs of labiodentals) and descend again until *lv* (pairs

of labiodentals and velars), rise again at *aa* (pair of alveolars) until *av* is reached (alveolars and velars), rise on *pp* (pairs of palatals), descend until *pv* (palatals and velars) and rise again on *vv* (pairs of velars). And that is indeed what mostly can be observed in the graph. The only exception is at *bv* where the ratings rise slightly, instead of decreasing.

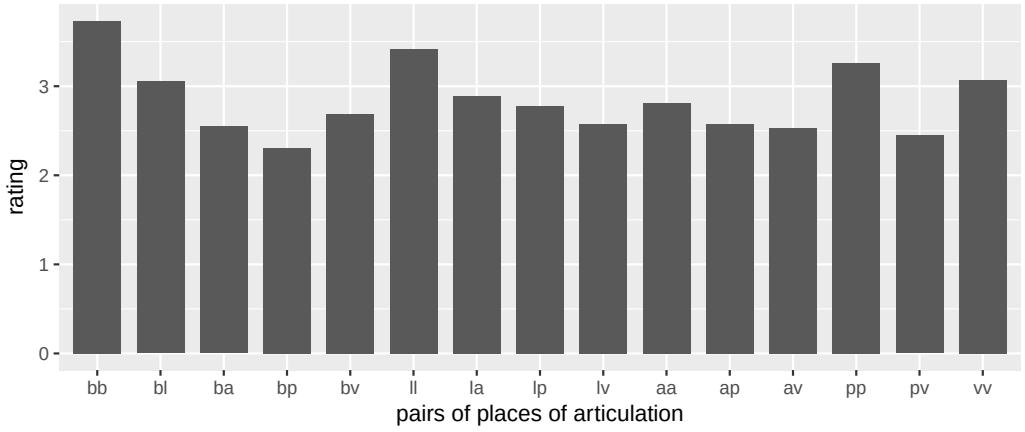


Figure 6: Average similarity ratings of consonants in different places of articulation. For example bb means that on average pairs of two bilabials were considered very similar.
b = bilabial, l = labiodental, a = alveolar, p = palatal, v = velar

Distance pair 1	bb	bl	ab	bp	bv	ll	al	lp	lv	aa	ap	av	pp	pv
Distance pair 2	bl	ab	bp	bv	ll	al	lp	lv	aa	ap	av	pp	pv	vv
p-value	**0.03**	**0.001**	0.09	**0.02**	**0.01**	0.06	0.43	0.21	0.09	**0.05**	0.63	**0.02**	**0.01**	**0.001**

Table 5: p-values for the ratings of pairs of places of articulation

Table 5 shows the p-values from a two-sample t-test for pairs of places of articulation. For example, for **bb-bl** $p = 0.03 < 0.05$ indicates that the similarity between two bilabials is significantly higher than the similarity between a bilabial and a labiodental consonant. Thus, for more than half of the bars in Figure 6 the difference between the ratings is significant.

In Figure 7 the ratings were averaged over all possible combinations of manners of articulation and sorted by the similarities that would be expected according to Theory 2, analogous to Figure 6. By degree of being consonantic, the manners are: glides (gl), liquids (li), nasals (na), fricatives (fr) and plosives (pl). It would be expected that glides are most sonorant/least con-

sonantic and that they have high similarity ratings with liquids, less similar ratings with nasals and that they are least similar to plosives. That means that pairs of glides and liquids (glli) would have a spike in the graph, while the bars would get shorter until hitting pairs of glides and plosives (glpl). Then the graph again would have a spike for pairs of liquids (lili) and decrease until it reaches the pairs of liquids and plosives (lipl). Same between nana (pairs of nasals) and napl (nasals and plosives), frfr (pairs of fricatives) and frpl (fricatives and plosives) and finally another spike for plpl (pairs of plosives) should be seen.

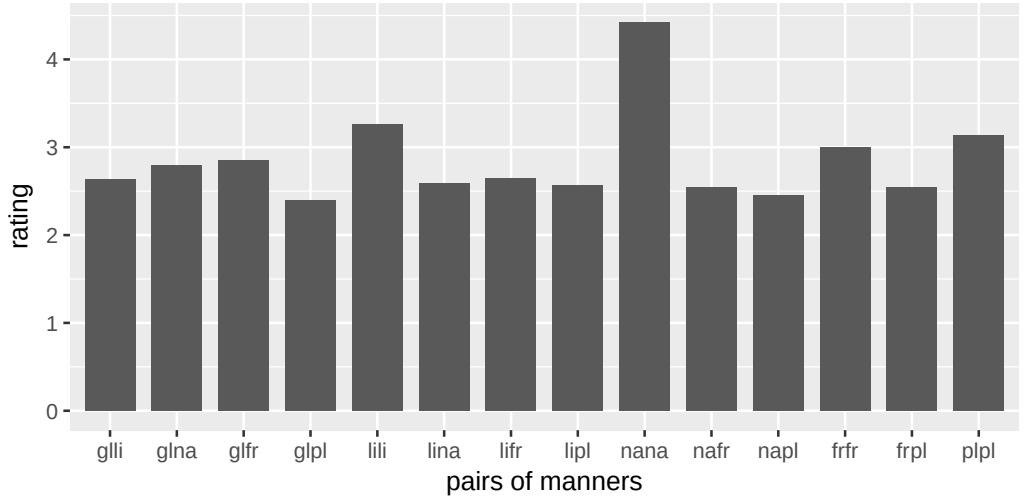


Figure 7: Average similarity ratings of consonants with different manners of articulation. For example nana means that on average pairs of two nasals were considered very similar.
gl = glides, li = liquids, pl = plosives, na = nasals, fr = fricatives

pairs 1	glli	glna	frgl	lili	lina	lifr	nana	nafr	frfr
pairs 2	glna	glfr	glpl	lina	lifr	lipl	nafr	napl	frpl
p-value	0.507	0.746	**0.005**	**0.035**	0.689	0.468	$\mathbf{1.35 \cdot 10^{-8}}$	0.433	$\mathbf{5.0 \cdot 10^{-5}}$

Table 6: p-values for the ratings of pairs of manners of articulation

Figure 7 only partially shows what would be expected according to Theory 2. While one would expect glli to have a spike, it has a relatively low value. Instead of decreasing, the pairs of manner increase between glli and glfr. Only some (expected) spikes can be seen at lili, nana, frfr and plpl which would

denote the most similar pairs. Most values in between the spikes do not show any significant rating differences, as can be seen in Table 6.

6.7.3 Linear Mixed Models

Linear Mixed Models offer a framework which can help us to look for correlates in the data. The relationship between different variables can be expressed as

$$y = X\beta + Zu + \varepsilon$$

y being a vector of observations of the dependent variable (the thing to analyze), X being a matrix of fixed effects (the things understood), Z being a matrix of random effects, ε being a vector of errors and β and u being vectors of coefficients connecting y to X and Z. A LMM can also be interpreted as a prediction of the dependent variable through fixed and random effects.

Analogously the *rating* for a syllable pair can be used as a dependent variable and *manner* and *location* as independent variables. If there is a correlation between rating, location and manner, location and manner should be helpful in predicting the rating. It can also be assumed that the ratings can differ between participants (for example there were participants who considered almost all syllable pairs to be very dissimilar). Therefore, the participants can be taken as a random effect.

R is used to calculate Linear mixed models for the data mentioned above (R Core Team, 2015). Libraries used for the analysis were *reshape2* and *lme4*.

The model looks as follows:

$$rating \sim manner + location + (1|participant)$$

Table 7 shows the coefficients of this model. The *Estimate* of the *Intercept* corresponds to the average *rating* for the similarity of the syllable pairs, while the estimate of each of the different *manner* and *location* pairs indicates

	Estimate	Std. Error	t value
(Intercept)	3.10324	0.16223	19.129
manner_frgl	0.01582	0.15236	0.104
manner_frli	-0.37729	0.11449	-3.295
manner_frna	-0.40598	0.12959	-3.133
manner_frpl	-0.44916	0.16176	-2.777
manner_glli	-0.06838	0.19727	-0.347
manner_glna	0.05557	0.20258	0.274
manner_glpl	-0.33705	0.20902	-1.613
manner_lili	0.15992	0.26186	0.611
manner_lina	-0.35666	0.16991	-2.099
manner_lipl	-0.30846	0.17995	-1.714
manner_nana	1.62676	0.27988	5.812
manner_napl	-0.51899	0.19758	-2.627
manner_plpl	0.13013	0.25627	0.508

	Estimate	Std. Error	t value
(Intercept)	3.10324	0.16223	19.129
location_ab	-0.14903	0.15936	-0.935
location_al	0.01834	0.11410	0.161
location_ap	-0.40328	0.12867	-3.134
location_av	-0.30895	0.13608	-2.270
location_bb	0.50347	0.33565	1.500
location_bl	0.39855	0.18392	2.167
location_bp	-0.40750	0.19238	-2.118
location_bv	-0.33279	0.23123	-1.439
location_ll	0.31781	0.26186	1.214
location_lp	-0.33483	0.17535	-1.910
location_lv	-0.08953	0.15392	-0.582
location_pp	0.14410	0.29058	0.496
location_pv	-0.33540	0.16254	-2.064
location_vv	0.26955	0.23123	1.166

Table 7: Coefficients for fixed effects of
$rating \sim manner + location + (1|participant)$

an increase in *rating*. This would for example mean that the rating for the difference in manner between fricatives and nasals is on average $3.10324 + (-0.40598) = 2.69726$. Following common practice it is assumed that an effect is significant if $2 < |t|$. Here, the *t-value* of $2 < |t| = 3.133$ is a first indicator that the effect is significant. It can be seen that for manner almost half of the t-values are bigger than 2, while for place only five out of 14 are bigger than 2. This suggests that manner is more important than place for the perception of rhyme quality.

To test whether the effect of *manner* and *location* is indeed significant, additionally the models with and without *manner* and/or *location* can be compared by ANOVAs as suggested by Winter (2013). The results of the ANOVAs show that a model with either *manner* or *location* provides a better fit. Furthermore, a model with both, *manner* and *location*, provides a better fit than a model considering only one of the features. According to Burnham and Anderson (2003) the difference between the AICs of the models (ΔAIC) is an important indicator for the goodness of fit. The lower AIC is better if $\Delta AIC > 2$. Table 8 shows the $\chi^2(1)$, p and ΔAIC values of all model combinations that were considered.

Model	$\chi^2(1)$	$\mathbf{p-value}$	$\mathbf{\Delta AIC}$
$score \sim manner + location + (1\|participant)$ $score \sim (1\|participant)$	199.81	$2.2 \cdot e^{-16}$	145.8
$score \sim location + (1\|participant)$ $score \sim (1\|participant)$	89.894	$3.975 \cdot e^{-13}$	61.9
$score \sim manner + (1\|participant)$ $score \sim (1\|participant)$	124.59	$2.2 \cdot e^{-16}$	98.6
$score \sim manner + location + (1\|participant)$ $score \sim manner + (1\|participant)$	75.218	$2.159 \cdot e^{-10}$	47.2
$score \sim manner + location + (1\|participant)$ $score \sim location + (1\|participant)$	109.92	$2.2 \cdot e^{-16}$	83.9

Table 8: ANOVA results for Linear Mixed Models

6.7.4 Discussion

The evidence supports both theories only in parts. While there are significant differences between some pairs of manners of articulation and between pairs of locations of articulation, a lot of the differences are not significant. With more participants the experiment might yield more significant results. The graphs indicate that both theories may be true and correlated. Particularly striking are the similarity ratings for consonant pairs which are produced in the same manner and in the same place.

Indicators for and against both theories can be seen. While for manner in Figure 7 it can be partly seen that the ratings agree with the theory, in Table 4 it can be seen that even for consonant pairs produced in the same manner, the ratings can be quite low (see [ʁ][ɕ], [ʁ][s],[s][ɕ]). As two of them are pairs of voiced and unvoiced consonants of the same manner, the difference between voiced and voiceless consonants could be a crucial factor. Especially as [ʁ] and [ɕ] do not seem to be far apart in regards to their place of articulation and one would not expect the place of articulation to have much effect on the pairs in such cases.

Aside from one exception, Figure 6 is consistent with what would be expected of Theory 1. The further apart the production locations are, the less similar the pairs are. However, in Figure 4 [ʁ][l] and [n][ŋ] have a rather high rating

although they are relatively far apart, while [s][l] or [k][j] have a very low rating although they are close to each other.

The ANOVAs of the Linear Mixed Models provide evidence that indeed both, *manner* and *location*, are significant to predict whether a syllable pair is similar. The results however do not allow a more fine-grained prediction that can be generalized for all articulatory-based classes.

If acquiring more participants would not help to get more significant results, it would have to be concluded that there must be more effects involved in consonant similarity than just the ones analyzed:
[ŋ] is between [ç] and [ʁ]. Both [ç] and [ʁ] are fricatives so manner of production should not have influence on the similarity and neither should the place of production, as [ç] and [ʁ] are direct neighbors. However, the rating for [ŋ][ç] is rather high and for [ŋ][ʁ] it is rather low. One difference would be that [ç] is unvoiced, while [ʁ] is voiced. But voice is unlikely to be a huge factor if looking at the high ratings for pairs like [k][g] or [p][b] which are both pairs of voiced/unvoiced consonants. Another difference might be in the amount of friction in fricatives or different language dependent phonological rules that people apply so that certain consonants sound similar, although their underlying forms are not very similar (e.g. [ng]→[n]).

If too many effects have an influence on the similarity of consonants, it becomes pointless to try to group them. Instead, all consonants would have to be analyzed individually.

6.8 A general approach for vowel and consonant similarity

A seemingly more fine-grained approach that considers all dimensions is proposed by Manurung et al. (2008) who based their similarity metric on Ladefoged and Halle's (1988) hierarchical feature decomposition of IPA symbols. Ladefoged and Halle (1988) proposed to model sounds from the IPA chart

as a hierarchical structure (Figure 8). The hierarchy consists of three levels (without the root node). The first level distinguishes between vowels and consonants. For vowels, the second level consists of the three common vowel dimensions (roundness, backness and closeness), while for consonants the second level contains the consonant dimensions (manner, place, voicing).

Unisyn (Fitt, 2000) which was used in Manurung et al. (2008) is a pronunciation dictionary that encodes different English accents. In this dictionary one may find strings that encode different variants of the same sound. For example {@, @r, @@r} are different types of [ə] (short, middle, long). Therefore, the second level (Figure 8) is not enough to encode a unique sound. Only using the second level to encode @, would result in the features `Mid-Center-Unrounded` but they can not encode the different sounds from the example. In such ambiguous cases the third level features get a distinctive value added to the features of the second level. The third level features would look somewhat like this: `Mid-Center-Unrounded:@r`.

The weighting in Manurung et al. (2008) worked as follows: Each of the attributes was assigned a cost between 0 and 1. If Level 1 for two symbols was identical, then Level 2 was considered, and so on. The overall cost (or what is called distance in this thesis) is the sum of all costs of the attributes which were found to be different. For a dissimilarity on Level 1 (vowels versus consonants) the costs were 1 (maximal), as vowels and consonants are considered to be maximally dissimilar. In Level 2 consonant attributes had a cost of 0.28 and vowel attributes 0.15. Differences in Level 3 were 0.15 for both vowels and consonants.

Example: consider [k] and [v]. Both are consonants, so Level 1 is identical and Level 2 has to be considered. [k] has the features `Unvoiced-Velar-Stop`, [v] has the features `Voiced-Labiodental-Fricative`. As all three features are different and the costs for each dissimilarity in Level 2 are 0.28, it results in the overall costs of $0.28 + 0.28 + 0.28 = 0.84$.

As in the data given in this thesis all sounds can be defined distinctively by their usual three dimensions, Level 3 can be ignored. Furthermore, only vowel

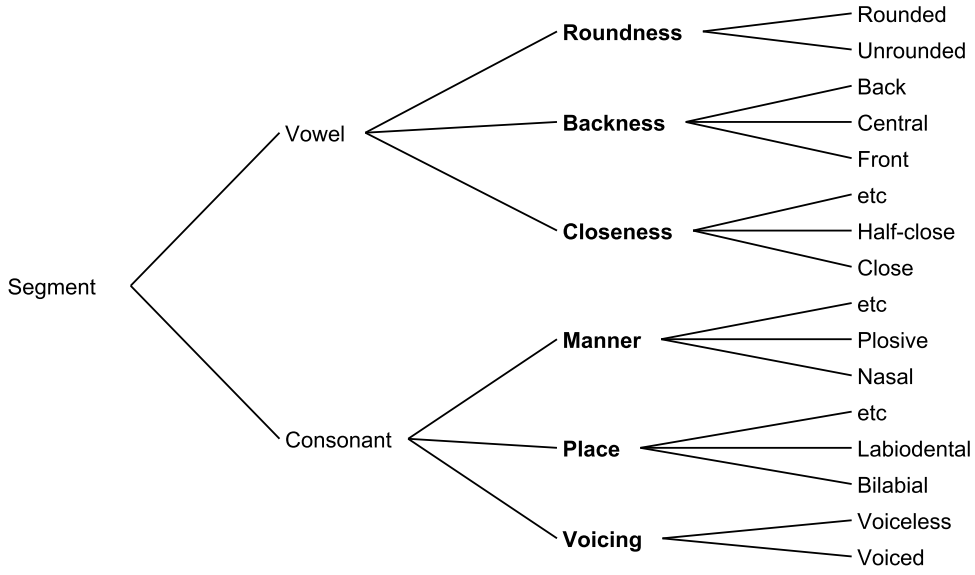


Figure 8: IPA categories as hierarchy, adapted from Ladefoged and Halle (1988)

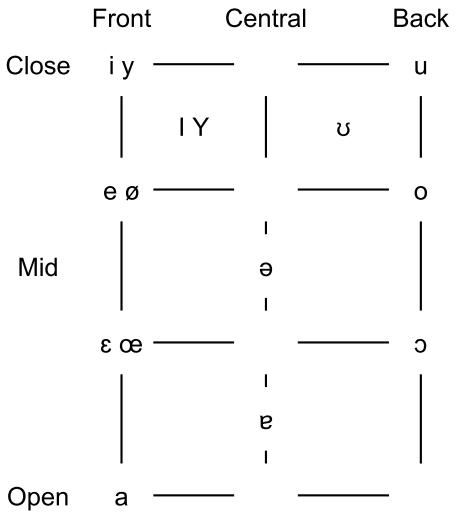
pairs and consonant pairs are compared. Consonant-vowel pairs are not compared, therefore Level 1 also can be ignored. An adapted metric that could be used in this thesis would simply have the same distance between all three dimensions in Level 2 for both vowels and consonants. For example, the pair [p]/[ʁ] would be different in manner (plosive/fricative), place (bilabial/uvular) and voicing (unvoiced/voiced) and as dissimilar as two consonants can get in that model. This would yield a scale of four different similarity levels: Either all three attributes are the same (then the consonant pair is identical) or two out of three attributes are the same or one or none. This metric corresponds to the best-performing metric as proposed by Bailey and Hahn (2005), where only the basic features were considered in measuring the distances between phonemes. Essentially, this is the vowel similarity described in 3.2 Approach 1, with the restriction that a vowel pair with a difference in more than one attribute is not considered similar anymore.

6.9 Considerations about vowel metrics

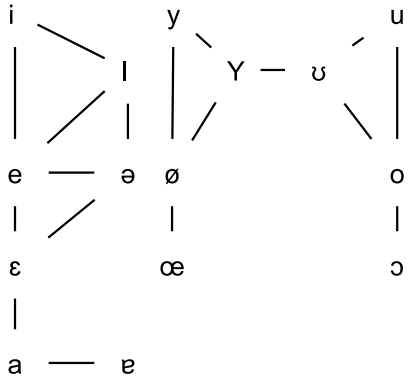
In 6.2 possible metrics for measuring vowel similarity were discussed. The organization of the vowels in German, as repeated in Figure 9a however, bears some problems. It seems reasonable to say that a small difference in the backness or in the closeness has a small enough difference in sound to make a syllable rhyme. The third dimension however, roundness, has a difference that is too extreme. Vowel pairs such as [i]/[y] and [ɪ]/[ʏ] have the property that one vowel is maximally unrounded while the other is maximally rounded. There are no steps in between, such as half rounded vowels. Therefore, it is questionable as how similar they should be considered. The approach taken in this paper is to allow only vowel pairs that are identical in their roundness property. This means that the pairs [ɪ]/[ʏ] and [i]/[y] are not considered to be similar.

Furthermore the vowels [ɪ], [ʏ] and [ʊ] have no edges connecting them to the other vowels. In these cases they are connected to their nearest vowel with the same roundness. Accordingly, [ʊ] is considered similar to [u] and [o], while [ɪ] is considered to be similar to [i] and [e] but not to [y] or [ʏ]. The next rounded vowel to [ʏ] is [ʊ], which also is not too far away on the backness scale. As there is no similarity on the roundness scale, the vowel graph splits in two parts. In the bottom part in a typical vowel diagram the vowels are closer together, as the backness distance for vowels decreases with higher openness due to converging formants between the vowels. This allows to connect vowels like [a] and [ɐ] and even [e] and [ə].

From these considerations a corresponding graph can be deduced, which is seen in Figure 9b. The positions of the sounds in the graph in a way represent their three dimensions (unrounded vowels on the left, rounded vowels on the right; close to open; front to back). However, considering the metric that is used to calculate similarity, the positions are irrelevant; it is only important which neighbors a vowel has.



(a) German vowel diagram



(b) Similarity graph for vowels in German. Rounded and unrounded vowels are separated.

Figure 9: Similarity graph for vowels in German

7 Implementation

The previous section described ways to measure the similarity of syllables by means of measuring the similarity of nuclei and codas. This section describes in detail how rhyme search was implemented.

7.1 Corpus

The corpus consists of several books in different genres, which were retrieved from the German section on `gutenberg.org`. License and disclaimer were pruned, as these were written in English. The corpus consists of 23 random books and approximately 1.2 million words. More would be possible, however with the current implementation a search is rather time consuming and the corpus size allows to retrieve good results with an even smaller corpus anyway.
Some of the books used were: **Schelmuffskys wahrhaftige, kuriöse und sehr gefährliche Reisebeschreibung zu Wasser und zu Lande** by **Christian Reuter**, **Herr und Knecht** by **L. N. Tolstoi** or **Buddenbrooks - Verfall einer Familie** by **Thomas Mann**.
The text was preprocessed, as the grapheme-to-phoneme converter could not handle certain characters. Then the text was tokenized and transcribed into SAMPA using Festival (v1.95) in order to be able to compare the sounds.
The g2p converter returns the transcriptions divided into syllables, which allows to analyze the text syllable-wise. Spaces between syllables are taken into account, although for a rhyme this is irrelevant, as a rhyme can consist of parts of words (Relativitäts**theorie erfunden**/Schiffsuntergangs**melodie erklungen**). Punctuation was ignored, although a syllable sequence could be penalized if it had a period inside of it.

Example:

Das ist ein Beispielsatz. Dies ist ein Beispiel.

Would be transcribed as:

["das]_["Ist]_["aIn]_[baI]["Spi:l][zats]_["di:s]_["Ist]_["aIn]_["baI][Spi:l]

7.2 Metric for Codas

As described in 6.2, vowel similarity can be measured by analyzing their distances in the vowel diagram or by comparing Euclidean distances between formants. Both approaches were implemented.
For single consonants, as explained in 6.6, the similarity can be measured by comparing their manner or their location. As in literature ranking consonants by manner seems to be the prevalent way, this is how it was implemented.
For sequences of vowels and consonants these metrics need to be combined. The following two sections describe how this was done.

A common way to compare two strings is to compute the Levenshtein distance. The algorithm analyzes the strings for three different operations, namely whether an insertion, a replacement or a deletion happened in each of the characters during the transformation from one string to another. Depending on which operation happened, the costs can vary. In case of codas in rhymes, this means that insertion and deletion should have the same cost, as it does not make a difference if coda a is transformed into coda b by inserting a consonant (run $\rightarrow$ runs) or coda b is transformed into coda a by deleting a consonant (runs $\rightarrow$ run). The rhyme run/runs is as good (or bad) as the rhyme runs/run. Substitution is the only operation that needs extended analysis: According to Berg (1998) (as shown in 6.6) the similarity of two consonants is dependent on their sonority. While [p] and [b] are very similar, [p] and [s] are less similar and [p] and [j] are very dissimilar. To accommodate the different distances between consonants, they were grouped by their manner of production and sorted by their sonority. Each group was

assigned a value, according to its sonority as shown in Table 9. When the Levenshtein algorithm compares two consonants and decides to replace one consonant by another, the normalized distance between the values of the two consonants' groups is used as the substitution value (see Listing 1, lines 17-21). For example, replacing [j] by [v] costs $\frac{5-1}{7} = 0.571$, as [j] has the value 1, [v] has the value 5 and the difference in the values is normalized by $N-1$, N being the number of different manners of articulation. This way the modified Levenshtein distance quantifies the similarity between two codas, considering different similarity distances between consonants with different manners of production. In addition, it is assumed that replacing a consonant is bad enough to penalize every replacement by 1. Therefore, the overall costs for replacing [j] by [v] would be $1 + \frac{5-1}{7} = 1.571$.

To sum up, the distance of two consonants is calculated as

$$D_{(c_1,c_2)} = \begin{cases} 0 & \text{if } c_1 = c_2 \\ 1 + \frac{|value(c_1) - value(c_2)|}{N-1} & \text{else} \end{cases},$$

where D_{c_1,c_2} is the distance between the two consonants c_1 and c_2, $value(c_1)$ and $value(c_2)$ are the values assigned to each manner of production that the consonants are elements of and N is the number of manners.

The distance between two codas is calculated as the sum of all distances of the consonants in the coda.

$$D_{coda} = \sum_{(c_{n_1},c_{n_2})} D_{(c_{n_1},c_{n_2})}$$

Manner	**gl.**	**rho.**	**lat.**	**nasals**	**v. fr.**	**unv. fr.**	**voiced pl.**	**unvoiced pl.**
Cons.	j, w	r	l, ʟ	m, ŋ, n	ç, ʁ, v, ʒ, z	f, h, ʃ, s, θ, x	b, d, g	ʔ, k, p, t
Value	1	2	3	4	5	6	7	8

Table 9: Consonants grouped by manner of production and assigned values

Examples:

1. Assume **Ring** (/ʀiŋ/) and **String** (/stʀiŋ/) which is a perfect rhyme and **Ring** and **Rind** (/ʀint/) which could be considered a slant rhyme. Only analyzing the coda, the distance between the [ŋ] in the perfect rhyme is 0, as they are identical. So the modified edit distance will be 0. [ŋ] and [t] in the slant rhyme however, are produced in different manners. According to Table 9 [t] has the value 8 and [ŋ] has the value 4. Therefore, the distance for the codas is $1 + \frac{8-4}{7} = 1.571$.

2. Assume **Strand** (/ʃtʀant/) and **Brand** (/bʀant/) as a perfect rhyme and **Strand** and **Rang** (/ʀaŋ/) as a slant rhyme. Again, since the perfect rhyme has identical codas the distance between the codas is zero. For the slant rhyme /nt/ is transformed into /ŋ/[2]. Here the algorithm first replaces the [n] by an [ŋ]. As both consonants are nasals and therefore are assigned the same value (4), the penalty for this replacement defaults to 1. In a second step [t] is deleted which again has the cost of 1. Therefore, the overall distance for the two codas is $1 + 1 = 2$.

7.3 Algorithm

The basic algorithm is able to find rhymes for an input word or word sequence, considering (Figure 11) or disregarding (Figure 12) word boundaries.

In a first step syllable sequences are gathered from the corpus which have the same number of syllables as the input word. From these, the syllable sequences with spaces in between can easily be filtered, to find only syllable sequences that are single words, in case it is desired.

The nuclei in each of these sequences are extracted and compared to the nuclei in the input word. If the nuclei of the syllables in the input word are

[2]Saying that /ŋ/ is transformed to /nt/ would be equivalent. In this case [t] would be inserted instead of being deleted but it would have the same costs.

Listing 1: Modified Levenshtein distance

```
coda_distance(s1, s2){
  if length(s1) < length(s2) {
    return coda_distance(s2, s1)
  }
  if length(s2) == 0 {
    return length(s1)
  }
  for char1 in s1 {
    for char2 in s2 {
      # same values for deletions and insertions
      insertions = previous_row[next_index] + 1
      deletions = current_row[current_index] + 1
      if c1 == c2 {
        subst_value = 0
      }
      else {
        # difference between group values of two characters
        value_distance = char1_value - char2_value
        # normalize distance between consonants
        #by number of group values (= number of manners)
        subst_value = (distance/num_manners) + 1
      }
      substitutions = previous_row[current_index] + subst_value
      current_row.append(min(insertions, deletions, substitutions))
    }
    previous_row = current_row
  }
  return previous_row[-1]
}
```

hin**zu**ziehn/hin**ter**liehn

/hɪntsuːtsiːn/
/hɪntɐliːn/

Figure 10: Triple rhyme with center syllable pair that does not rhyme

not similar or identical to the corresponding nuclei in an extracted syllable sequence, the sequence is discarded from future calculations. As vowels (i.e. nuclei) are the most important sounds in a rhyme (Menon, 2006), it makes little sense to consider syllables with nuclei that are too different. The nuclei are extracted as described in Approach 1 (6.2) or Approach 2 (6.2).

The syllables in the remaining syllable sequences are split into nuclei and codas. Each nucleus pair is assigned a distance using the Formulas 1 and 2, based on the similarity graph in 9b or using the mel-distances, as described in Approach 2 (3.2). For each coda pair the modified Levenshtein distance is computed, as described in 7.2.

Another constraint was added to the algorithm: The author perceives rhymes as single entities or *units*. These units have a beginning and an end. What is inside the rhyme seems to be less important (but not unimportant). Therefore, the rhyme in 10 is perceived as a triple rhyme instead of two single rhymes, although only the first and the last syllables rhyme.

The same effect can be observed in phrases. Typically, one develops an expectation for the last phrase or the last word to rhyme if the word has many syllables. If the first and the last syllables in two units clearly separate the syllable sequences from the text, the units are perceived as rhymes and can bear missing some rhyming syllables in between, without sacrificing too much rhyme quality. This is why it makes sense to be sure that the first and last syllable are “good” rhymes. Thus, the algorithm looks only for syllable sequences where syllable pairs in the first and in the last syllables have vowels that match exactly. Diphthongs must contain identical vowels.

Massenvernichtungswaffen/Knaben der sich vor lachen

/ma-sən-fɛɐ-nɪç-tuŋs-va-fən/
/kna-bən deɐ ziç foɐ la-χən/

Figure 11: Rhyme without considering word boundaries

The distances of all nucleus and coda pairs between the input word sequence and the found sequences in the corpus are then summed, providing a means to measure the rhyme quality. A perfect rhyme would have a score of 0, while a bad rhyme would have a high score. s_{max}, the highest score possible, can be computed as

$$s_{max} = d_{vmax} \cdot s + m \cdot d_{cmax} \cdot s$$

where d_{vmax} is the maximum distance between two vowels, s is the number of syllables, d_{cmax} is the highest distance between two consonants and m is the maximum number of consonants that can occur in a coda. The equation does not consider the constraint on the first and last nucleus which only allow the morae to be different. For a rhyme with four syllables in German the highest score possible would be

$$2.5 \cdot 4 + 2 \cdot 5 \cdot 4 = 50,$$

as the highest score for a vowel (or diphthong) is 2.5, the highest score for a consonant is 2 (according to 7.2) and the maximum number of consonants in a coda in German is 5 (Graefen and Liedke, 2012).

7.4 Strengths and weaknesses of the implementation

In measuring vowels as well as consonants, the features used are mostly assumed to have equal importance. For open vowels in **Apprach 1** (6.2) the

Konditionieren/Konstituieren

/kɔn-di-tsjo-ni-ʀən/
/kɔn-sti-tu-i-ʀən/

Figure 12: Rhyme with considering word boundaries

smaller distance in the backness feature was taken into account, allowing to increase the number of similar neighbors. Rounded and unrounded vowels were also divided due to the lack of steps in between. However, this does not allow conclusions regarding the amount of influence of one feature on the similarity of two vowels. There could be differences in the prominence or *salience* (Kondrak, 2002) of a feature due to the amount of changes in the vocal tract or due to the resulting frequencies. The difference between a closed and an open vowel may have a small (or big) difference in sound, despite (or due to) the amount of change in the vocal tract, compared to the difference between rounded and unrounded vowels. Although others came up with ways to measure the salience of features, for example using multivalued phonological features (Kessler, 1995), assigning them ordinal values between 0 and 1 or using binary features (Nerbonne and Heeringa, 1997), it is difficult to get a representative scale. When calculating a similarity of two sounds the features must be assigned saliences which are probably inaccurate for many reasons, such as different languages, dialects or mutual influence of features on each other. To test whether the right features were picked and whether they were weighted correctly, a gold standard would be needed, such as a well annotated corpus of (rap) lyrics. As rhyme quality is highly subjective and also depends on experience, no gold standard can exist to compare rhymes by their goodness. The only other way would be to create a database using the same - probably inaccurate - assumptions to test rhymes against it.
Computing vowel similarity by only considering formants unveils similar problems. While the three dimensions of backness, openness and roundedness are implied in the formants, these and other features, such as morae interact

with different formants to different degrees. There is no one way to define the salience of the formants or features. Changing the parameters may change the results dramatically.

For consonants, *manner* seems to be more commonly used to determine similarity than *place*, according to prevailing literature (Croft, 2002; Berg, 1998; Kawahara, 2007). This indicates that manner as a feature is more salient than place. The experiment conducted in this thesis to compare both theories provides (weak) evidence for the opposite if considering simple t-tests, though it provides (weak) evidence for the results in literature if looking at the Linear Mixed Models results. Again, it is not clear how much more salient manner or place are.
According to the decline in consonanthood as proposed in Croft (2002), phonation is only important in a narrow area of the consonanthood scale; Voiced and voiceless stops, as well as voiced and voiceless fricatives are considered to be very similar. It is not clear whether phonation or place is more important and how much more important one is compared to the other in relation to manner.

It shall be noted that the features that are used are articulatory features that might be more relevant in *speech production* tasks, whereas the task at hand is to output syllable sequences which are *perceived* as rhymes. However, what is perceived is highly dependent on how it is produced and by whom.

A general advantage of using text corpora for the extraction of rhyming word sequences is that syntactical nonsense is automatically avoided, as opposed to randomly combining different words in a lexicon to form rhymes.

The complete algorithm can be found in the Appendix.

8 Results and Discussion

In the following results for some test words are discussed and the two approaches to score vowels are compared.

Listing 2 shows five best out of several hundred results for the rhymes found for **Massenvernichtungswaffen**. Selecting these rhyme candidates was done by the method described in **Approach 2** (6.2). The first brackets `['alles', 'fertig', 'so', 'packte']` show the results found in clear text, the second brackets show the result as a SAMPA transcription.
The next line shows the scores for this rhyme candidate. The first brackets represent the scores for the nucleus pairs. For example for **a** (`a`) and **lles** (`l@s`)[3] the vowels `a` are identical and therefore return a score of 0 for the first four nuclei. The fifth value has a mel-distance of 149. After mapping the value according to **Approach 2** in 6.2 the fifth value has a score of 1.86. In the second brackets the sum of all nucleus pair scores is shown. The third brackets show the scores for the codas. For the first coda pair the distance is 0.0 as both codas are empty. The second coda pair has a score of 1.29 as `C` was replaced by `n`. The fourth bracket shows the sum of all consonant distances. At the end the sum of all vowel and consonant scores (7.15) is shown.

Listing 3, again, shows five best out of several hundred results for the rhymes found for **Massenvernichtungswaffen**. Selecting these rhyme candidates was done by the method described in **Approach 1** (6.2). While the consonants are computed in the same manner as described before, the similarities between vowel pairs are computed by looking at their neighbors in the vowel diagram. For the first result the first syllable gets a score of 1.5 as the vowels in `kna:` and `ma` are different (+1) and have different morae (+0.5).

With higher number of syllables the scores increase severely. While there is a good chance to get rhymes with low scores when querying three or four syllables, it is unlikely to get a low score for a query that has seven syllables, like **Massenvernichtungswaffen**. However, **Knaben der sich vor lachen** in 3 appears to be a very good result, even though it has a score of 6.5. Often such a high score for shorter rhymes is not even recognizable as a rhyme

[3]The grapheme to phoneme converter produced a hyphenation error. Correct would be **al·les**, instead of **a·les**. However, it does not have an impact on the rhyme quality.

anymore. In this case the nucleus pair `ma` and `kna:` has a comparably high score, due to having different morae and being different vowels, however they appear to rhyme almost perfectly. Furthermore, for the syllable pair `de:6` and `fE6` the nucleus pairs got a score of two due to `E6` and `de:6` being two different diphthongs. For diphthongs a lot of cases are not accounted for, such as a diphthong with a long first vowel which occurs rarely. In this case it was assumed that one of the nuclei is a diphthong and the other is not. Thus, the score should be smaller. On the other hand one of the initial vowels is longer, thus it would make sense to penalize this fact. The main problem however seems to be that both vowels sound like a perfect rhyme and a score of 0 feels most appropriate. These disparities between subjective (personal) opinions and "objective" (calculated) ratings illustrate that there is a lot of room for tuning the algorithm.

The first result in Listing 5 exemplifies a topic that was not discussed in this thesis. **Silbendurchfall** and **bilden sobald** is a rather bad rhyme at first glance. While **silben**/**bilden** rhymes well, **durchfall**/**sobald** does not seem to rhyme at all. This is because stress was never considered. It appears to rhyme better if for both words the same syllable positions are stressed, e.g. (**dúrchfall**/**sóbald**). In 7.3 the effect was described that in a rhyming unit the negative effects of a non-rhyming syllable are diminished if the non-rhyming syllable is surrounded by rhyming syllables. This effect can be observed here: **Silbendúrchfall**/**bilden sóbald** still sounds like a slant rhyme, yet it appears to be a rhyme with four syllables. In practice the odd sounding stress in **sóbald** can be used as part of the flow, creating the impression that the sound is intended.

In Listing 4 all the results shown have a lot of function words. Rhymes can also be rated using their POS-tags. In general one could say that noun rhymes improve the overall impression of a rhyme over adjectives, over verbs over function words. Therefore, it might be useful to add POS penalties to the algorithm.

Listing 2: Results for “Massenvernichtungswaffen” based on mel-distances

```
INPUT: ['"ma', 's@n', 'fE6', 'nIC', 'tUNs', 'va', 'f@n']

['alles', 'fertig', 'so', 'packte'] [['"a', 'l@s'], ['"fE6', 'tIC'], ['"zo:'], ['"pak', 't@']]
[0.0, 0.0, 0.0, 0.0, 1.86, 0.0, 0.0] [1.86] [0.0, 1.29, 0.0, 0.0, 2.0, 1.0, 1.0] [5.29] 7.15
['ganzen', 'Berg', 'herummachte'] [['"gan', 'ts@n'], ['"bE6k'], ['"he:6', 'U', 'max', 't@']]
[0.0, 0.0, 0.0, 0.72, 0.0, 0.0, 0.0] [0.72] [1.0, 0.0, 1.0, 1.0, 2.0, 1.0, 1.0] [7.0] 7.72
['Schlange', 'jählings', 'zusammen'] [['"Sla', 'N@'], ['"jE:', 'lINs'], ['tsu:', '"za', 'm@n']]
[0.0, 0.0, 0.96, 0.0, 1.66, 0.0, 0.0] [2.62] [0.0, 1.0, 0.0, 2.14, 2.0, 0.0, 0.0] [5.14] 7.76
['Dampferverbindung', 'dachte'] [['"dam', 'pf6', 'fE6', 'bIn', 'dUN'], ['"dax', 't@']]
[0.0, 2.15, 0.0, 0.0, 0.0, 0.0, 0.0] [2.15] [1.0, 1.0, 0.0, 1.14, 1.0, 1.0, 1.0] [6.14] 8.29
['reisefertig', 'und', 'packte'] [['"RaI', 'z@', 'fE6', 'tIC'], ['"Unt'], ['"pak', 't@']]
[3.14, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] [3.14] [0.0, 1.0, 0.0, 0.0, 2.29, 1.0, 1.0] [5.29] 8.43
```


Listing 3: Results for “Massenvernichtungswaffen” based on vowel diagram

```
INPUT: ['"ma', 's@n', 'fE6', 'nIC', 'tUNs', 'va', 'f@n']

['Knaben', 'der', 'sich', 'vor', 'Lachen'] [['"kna:', 'b@n'], ['"de:6'], ['"zIC'], ['"fo:6'], ['"la', 'x@n']]
[1.5, 0, 2, 0, 1, 0, 0] [4.5] [0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0] [2.0] 6.5
['Wagen', 'legen', 'und', 'wachen'] [['"va:', 'g@n'], ['"le:', 'g@n'], ['"Unt'], ['"va', 'x@n']]
[1.5, 0, 2, 1, 0, 0, 0] [4.5] [0.0, 0.0, 0.0, 1.14, 2.29, 0.0, 0.0] [3.43] 7.93
['klagenden', 'Blick', 'musterten'] [['"kla:', 'g@n', 'd@n'], ['"blIk'], ['"mUs', 't6', 't@n']]
[1.5, 0, 2, 0, 0, 1, 0] [4.5] [0.0, 0.0, 1.0, 1.43, 1.0, 0.0, 0.0] [3.43] 7.93
['Sachen', 'alle', 'zusammen'] [['"za', 'x@n'], ['"a', 'l@'], ['tsu:', '"za', 'm@n']]
[0, 0, 2.5, 1.5, 1, 0, 0] [5.0] [0.0, 0.0, 0.0, 1.0, 2.0, 0.0, 0.0] [3.0] 8.0
['alle', 'wenig', 'zu', 'sagen'] [['"a', 'l@'], ['"ve:', 'nIC'], ['"tsu:'], ['"za:', 'g@n']]
[0, 0.5, 2, 0, 1, 1.5, 0] [5.0] [0.0, 1.0, 0.0, 0.0, 2.0, 0.0, 0.0] [3.0] 8.0
```

Listing 4: Results for “Silbendurchfall” based on mel-distances

```
INPUT: ['"zIl', 'b@n', 'dU6C', 'fal']

['mitten', 'durch', 'das'] [['"mI', 't@n'], ['"dU6C'], ['"das']]
[0.0, 0.0, 0.0, 0.0] [0.0] [1.0, 0.0, 0.0, 1.43] [2.43] 2.43
['Schlingen', 'durch', 'das'] [['"SlI', 'N@n'], ['"dU6C'], ['"das']]
[0.0, 0.0, 0.0, 0.0] [0.0] [1.0, 0.0, 0.0, 1.43] [2.43] 2.43
['Schritten', 'durch', 'das'] [['"SRI', 't@n'], ['"dU6C'], ['"das']]
[0.0, 0.0, 0.0, 0.0] [0.0] [1.0, 0.0, 0.0, 1.43] [2.43] 2.43
['Bitte', 'durch', 'das'] [['"bI', 't@'], ['"dU6C'], ['"das']]
[0.0, 0.0, 0.0, 0.0] [0.0] [1.0, 1.0, 0.0, 1.43] [3.43] 3.43
['blitzten', 'durch', 'das'] [['"blIts', 't@n'], ['"dU6C'], ['"das']]
[0.0, 0.0, 0.0, 0.0] [0.0] [2.43, 0.0, 0.0, 1.43] [3.86] 3.86
```


Listing 5: Results for “Silbendurchfall” based on vowel diagram

```
INPUT: ['"zIl', 'b@n', 'dU6C', 'fal']

['bilden', 'Sobald'] [['"bIl', 'd@n'], ['zo:', '"balt']]
[0, 0, 2, 0] [2] [0.0, 0.0, 1.0, 1.0] [2.0] 4.0
['finden', 'Rotang'] [['"fIn', 'd@n'], ['"Ro:', 'taN']]
[0, 0, 2, 0] [2] [1.14, 0.0, 1.0, 1.14] [3.28] 5.28
['dichten', 'Urwald'] [['"dIC', 't@n'], ['"u:6', 'valt']]
[0, 0, 2, 0] [2] [1.29, 0.0, 1.0, 1.0] [3.29] 5.29
['ich', 'den', 'Vorfall'] [['"IC'], ['"de:n'], ['"fo:6', 'fal']]
[0, 1, 2, 0] [3] [1.29, 0.0, 1.0, 0.0] [2.29] 5.29
['sitzen', 'Sobald'] [['"zI', 'ts@n'], ['zo:', '"balt']]
[0.5, 0, 2, 0] [2.5] [1.0, 0.0, 1.0, 1.0] [3.0] 5.5
```

9 Outlook

A lot of things can be done to improve the algorithm and to extend its functionality.
Further algorithm enhancements could involve tuning the parameters to achieve results that are more similar to human rating. While it may be argued that syllables not agreeing in stress could be avoided by adjusting the flow, a small cost penalty could still be considered. Many function words may trivialize a rhyme. Therefore, too many function words in a rhyme could be penalized too. As features on the vowel side the position in a vowel diagram or formants were examined. While both cases seem to deliver acceptable results, more features could be used. For example, by analyzing speech data, correlates between the given features and spectral tilt could be found, which may be helpful in improving the results.

On the consonant side, in this thesis either rating consonant pairs by manner or by location was considered. However, it is likely that both features matter. An elaborate assessment of this possibility could lead to improved, more realistic consonant metrics. Another approach could be to analyze consonant (and vowel) features according to the system proposed by Jakobson and Halle (1956). However, again, tuning the scores for all features is important, as the results could suffer severely by inconsiderate feature salience assignments.
Punctuation was ignored completely. While this may still be successfully implemented in a rap by means of flow, some results still would appear as syntactic nonsense. A restriction for syllable sequences which have no period in them could be made optional and/or be penalized.
The modified Levenshtein algorithm described in 1 assumes a distance of 1 for insertion and deletion, which essentially is a distance between "nothing" and a consonant. However, following the argument that vowels are the phonemes carrying the most information in a syllable, one could argue that the difference between /a/ and /al/ is much higher than between /a/ and /ap/, as more sonorant consonants are more prominent than less sonorant

consonants, because more sonorant consonants are “closer” to a vowel. Here, a principle to determine an insertion/deletion weight can be used that is similar to the way consonant similarity was estimated in 7.2. The formula $D = \frac{|value(c_1) - value(c_2)|}{N-1}$ could be used to determine a distance between nothing and a consonant by assigning 0 to one of the consonants, yielding a similarity score between 0 and 1.

While this thesis focused on retrieving multisyllabic rhymes, the range of functions could easily be extended to enable the program to identify other stylistic devices:

Homophones sometimes are used to replace a rhyme:

Dan Bull, The Crew Rap

Set the car onto **cruise control**
And roll on the roads that my **crews control**

For their rarity homophones may be considered by some as even more valuable than a good multisyllabic rhyme. An easy approach to find homophones would be to look for syllable sequences which are transcribed the same but written differently.

This approach would not work in a case such as the following:

You have a weak flow - stop bein’ a tough guy
You have a weak flow - go, see a doctor

In the ambiguous case of **You have a weak flow** (insinuating an urological issue, or the opponents problem delivering raps or insinuating that having a problem delivering raps could be treated by a doctor) the phrase is not only a homophone but it is also written identically. Here, comparing transcription and original text would not lead to success. Some kind of semantic disambiguation would be necessary, especially as the ambiguity is introduced by the textual context. Without context (or with only the wider context of battle rap) **You have a weak flow** simply would be understood as the statement that someone has a bad delivery.

Another stylistic device, alliteration, can easily be retrieved by filtering rhymes that don't begin with the same phoneme.

Double time rap is a rap performed twice as fast. If the beat has 80 BPM a rapper performs his lyrics as if the beat had 160 BPM. In order to rap with such a speed some lyricists write their texts in such a way that the articulators cover a distance which is as small as possible. This is especially easy when using many plosives in the frontal vocal tract area combined with mono- or disyllabic words.

In order to facilitate writing lyrics for double time rap, the application could be programmed to look only for syllables having phonemes which are in the vicinity in the vocal tract and filter for certain plosives.

The algorithm is easily modifiable to be used with any other language. Depending on the method that is used to find similarities between vowels or consonants, only the phonemes and their feature values available in the corresponding language have to be exchanged. For vowels, if a vowel diagram is used, the distances between the vowels have to be adapted. If mel-distances are used, the formants for the corresponding vowels need to be retrieved and converted. For consonants the corresponding consonants need to be arranged according to manner (or location). As morae are dependent on the language, the parameters here need to be adapted too.

10 Conclusion

The thesis showed a way to extract rhymes from a non-rhyming corpus. As opposed to classical rhyme search applications, the proposed method does not need a rhyme lexicon, (manual) annotation or a lyrical corpus. The only prerequisites needed are a grapheme-to-phoneme-converter and a random text corpus. To rate rhyme similarity, nuclei and codas were analyzed separately. To rate nucleus pairs by similarity, two methods were proposed: Using mel-frequencies or looking at neighbors of a nucleus in the vowel diagram. To estimate the similarity of coda pairs two methods were examined, namely to sort consonants by manner or by location. For that purpose an experiment was conducted but the results were inconclusive. Manner was chosen as the method to measure consonant similarity, as sorting consonants by manner seems to be prevalent in literature. All scores for vowel and consonant pairs were then summed to a final score. The results revealed that the algorithm indeed works very well but it also showed discrepancies between subjective ratings of the rhymes and the calculated scores for rhyme quality.

References

Karteek Addanki and Dekai Wu. *Unsupervised rhyme scheme identification in hip hop lyrics using hidden Markov models*. Springer, 2013.

Todd M Bailey and Ulrike Hahn. Phoneme similarity and confusability. *Journal of Memory and Language*, 52(3):339–362, 2005.

William H Baxter. *A handbook of Old Chinese phonology*, volume 64. Walter de Gruyter, 1992.

Thomas Berg. *Linguistic structure and change: An explanation from language processing*. Oxford University Press, 1998.

Kenneth P. Burnham and David Anderson. Model selection and multi-model inference. *A Practical information-theoretic approach. Springer*, 2003.

Helen Smith Cairns et al. *Fundamentals of psycholinguistics*. John Wiley & Sons, 2010.

Jeff Chang. *Can't stop won't stop: A history of the hip-hop generation*. Macmillan, 2007.

William Croft. *Typology and universals*. Cambridge University Press, 2002.

Sheila Davis. *The craft of lyric writing*. Writer's Digest Books, 1985.

INC Farlex. The free dictionary. *Retrieved August*, 15, 2016.

Festival. The Festival Speech Synthesis System. `http://www.cstr.ed.ac.uk/projects/festival/`, v1.95. Centre for Speech Technology, University of Edinburgh.

Susan Fitt. Documentation and user guide to unisyn lexicon and post-lexical rules. *Center for Speech Technology Research, University of Edinburgh, Tech. Rep*, 2000.

Gabriele Graefen and Martina Liedke. *Germanistische Sprachwissenschaft: Deutsch als Erst-, Zweit-oder Fremdsprache*, volume 8381. UTB, 2012.

T Alan Hall. *Phonologie: Eine Einführung*. Walter de Gruyter, 2011.

Hussein Hirjee and Daniel Brown. Using automated rhyme detection to characterize rhyming style in rap music. *Empirical Musicology Review*, 5 (4), 2010.

Hussein Hirjee and Daniel G Brown. Automatic detection of internal and imperfect rhymes in rap lyrics. In *ISMIR*, pages 711–716, 2009.

Roman Jakobson and Morris Halle. Fundamentals of language.'s-gravenhage: Mouton. 1956.

Keith Johnson. Higher formant normalization results from auditory integration of f2 and f3. *Attention, Perception, & Psychophysics*, 46(2):174–180, 1989.

Shigeto Kawahara. Half rhymes in Japanese rap lyrics and knowledge of similarity. *Journal of East Asian Linguistics*, 16(2):113–144, 2007.

Shigeto Kawahara and Kazuko Shinohara. The role of psychoacoustic similarity in japanese puns: A corpus study. *Journal of Linguistics*, 45(1): 111–138, 2009.

Brett Kessler. Computational dialectology in irish gaelic. In *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, pages 60–66. Morgan Kaufmann Publishers Inc., 1995.

Grzegorz Kondrak. Algorithms for language reconstruction. *PhD dissertation*, 2002. University of Toronto.

Peter Ladefoged and Morris Halle. Some major features of the international phonetic alphabet. *Language*, 64(3):577–582, 1988.

Ruli Manurung, Graeme Ritchie, Helen Pain, Annalu Waller, Rolf Black, and Dave O'Mara. Adding phonetic similarity data to a lexical database. *Language resources and evaluation*, 42(3):319–324, 2008.

Nina McCurdy, Vivek Srikumar, and Miriah Meyer. Rhymedesign: A tool for analyzing sonic devices in poetry. *Proceedings of Computational Linguistics for Literature*, 2015.

Mythri S Menon. *Foreign Accent Management*. Plural Publishing, 2006.

John Nerbonne and Wilbert Heeringa. Measuring dialect distance phonetically. In *Proceedings of the Third Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON-97)*, 1997.

Jaye Padgett. The unabridged feature classes in phonology. santa cruz, ca: University of california, ms, 2001.

Matthias Pätzold and Adrian P Simpson. Acoustic analysis of german vowels in the kiel corpus of read speech. *The Kiel Corpus of Read/Spontaneous Speech Acoustic data base, processing tools and analysis results. Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, 32:215–47, 1997.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL `https://www.R-project.org/`.

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: short papers-Volume 2*, pages 77–82. Association for Computational Linguistics, 2011.

John Simpson and Edmund SC Weiner. Oxford english dictionary online. *Oxford: Clarendon Press. Retrieved August*, 6, 2016.

Bodo Winter. Linear models and linear mixed effects models in R with linguistic applications. *arXiv preprint arXiv:1308.5499*, 2013.

Appendix

W

```
import os
import csv
import copy
from compiler.ast import flatten

vokale = r"IEaOUY69ieouy2@3"
test_words = [u'["bR2:t][x@n]',u'["tE:t]',u'[mo:][bi:][li:]["tE:t]',u'[an][tRo:][po:][lo:]["gi:]', u'[Un][fE6]["tsy:k][lIC]', u'[In][tE][lEk][tu:]["El]', u'[kOn][di:][tsjo:]["ni:][R@n]', u'["zIl][b@n][dU6C][fal]', u'[In][vEn][ta:][Ri:]["zi:][R@n]', u'["klaUs][tRo:][fo:][bi:]', u'["ma][s@n][fE6][nIC][tUNs][va][f@n]',u'[lIC]', u'[bi:]', u'["fROY][d@]']

# read consonanthood.txt used to find the distance between consonants
def readConsonantHood():
    touples = []
    with open("consonanthood.txt") as f:
        w = f.readlines()
        temp = [entry.strip("\n").split("\t") for entry in w]
        for entry in temp:
            del entry[0]
            touples.append(entry)
    return touples

# read mel distances
meldistances=[]
with open("meldistances.csv", "r") as f:
    reader=csv.reader(f, delimiter=',')
    for row in reader:
        meldistances.append(row)

# read vowels.txt used to find similar vowels
def readVowelMatrix():
    with open("vowels.txt") as f:
        w = f.readlines()
        touples = [entry.strip("\n").split("␣") for entry in w]
        return dict(touples)

# read transcribed sampa files
def readFiles():
    words = []
    for filename in os.listdir("Texte/Sampa"):
        with open("Texte/Sampa/" + filename) as f:
            w = f.readlines()
        words = words + w
    return words

# remove brackets
def strip_list(words):
    stripped_sampa = [words[pos][0].strip('[]').split("][") for pos in xrange(len(words)) if words[pos] != '']
    stripped_clear = [words[pos][1] for pos in xrange(len(words)) if words[pos] != '']
    joined = [i for x, y in zip(stripped_sampa,stripped_clear) for i in [[x, y]]]
    return joined

def strip_list_test(words):
    stripped = [words[pos].strip('[]').split("][") for pos in xrange(len(words)) if words[pos] != '']
    return stripped

```

```
54  # get n-th syllable of a word
55  def get_syl_at_pos(word, pos):
56      return word[pos]
57
58  #get last syllable of a word
59  def get_last_syl(word):
60      return word.split("][")[-1].strip('[]')
61
62  # get number of syllables in a word
63  def get_syl_num(word):
64      return len(word)
65
66  ############## vowels ##############
67
68  # get nucleus of n-th word
69  def get_nucleus(word, pos):
70
71      word_pos = word[pos]
72      for char_index, char in enumerate(word_pos):
73          if char in vokale:
74              try:
75                  next_char = ''
76                  next_char = word_pos[char_index + 1]
77                  if next_char == ":":
78                      return char + ":"
79                  elif next_char in vokale:
80                      return char + next_char
81                  else:
82                      return char
83              except:
84                  return char
85
86  # get the sequence of vowels in a word
87  def get_vowel_sequence(word):
88      vow_list = []
89      for syl_pos in range(get_syl_num(word)):
90          nucleus = get_nucleus(word, syl_pos)
91          vow_list.append(nucleus)
92      return vow_list
93
94  # get similar vowels for a word. if strict_vows is true, get only the same vowels
95  def get_similar_vowel_sequence(word, strict_vows):
96      syl_list = []
97      for syl_pos in xrange(get_syl_num(word)):
98          nucleus = get_nucleus(word, syl_pos)
99          nucleus = nucleus[0]
100         if strict_vows == True:
101             syl_list.append(vows[nucleus][0])
102         else:
103             vow_list = list(vows[nucleus])
104             vow_list_with_semicolon = [vow + ":" for vow in vow_list]
105             syl_list.append(vow_list + vow_list_with_semicolon)
106     return syl_list
107
108 # get vowel sequences from the corpus that are similar to the vowel sequence of the
        input word
109 def get_matching_vowel_sequence(words, test_word):
110     test_vowel_sequence = get_vowel_sequence(test_word)
111     matches = {word for word in words if get_vowel_sequence(word) == test_vowel_sequence
            }
112     return matches
113
114 # flattens a list with sublists, reducing the depth of the tree
115 def get_flat_list(nested_list):
```

```
116     return [item for sublist in nested_list for item in sublist]
117
118 #get word sequences of the same length as the test word
119 def get_word_sequences(words, test_word):
120     len_words = len(words)
121     max_syls = get_syl_num(test_word)
122     word_sequences = []
123
124     #for word_index in range(len_words):
125     for word_index, word in enumerate(words):
126
127         possible_words = []
128         nth_next_word = 1
129         possible_words.append(word)
130         curr_word_sequence_len = len(word[0])
131         next_word_index = word_index + nth_next_word
132
133         for _ in words[word_index:word_index + max_syls]:
134             if next_word_index < len_words:
135                 next_word = words[next_word_index]
136                 len_next_word = len(next_word[0])
137             nth_next_word += 1
138             next_word_index = word_index + nth_next_word
139             if curr_word_sequence_len < max_syls:
140                 curr_word_sequence_len += len_next_word
141                 possible_words.append(next_word)
142
143         if curr_word_sequence_len == max_syls:
144             word_sequences.append(possible_words)
145     return word_sequences
146
147 def get_reorder(word_sequence):
148     sampa = []
149     clear = []
150     for word in word_sequence:
151         sampa.append(word[0])
152         clear.append(word[1])
153     return [sampa]+[clear]
154
155 def get_unique_matching_similar_vowel_sequence(all_word_sequences):
156     unique=[]
157     for entry in all_word_sequences:
158         temp = entry[0]
159         count = 0
160         for unique_element in unique:
161             element = unique_element[0]
162             if element == temp:
163                 count += 1
164         if count == 0:
165             unique.append(entry)
166     return unique
167
168 # get word sequences with similar vowels
169 def get_matching_similar_vowel_sequence(word_sequences, test_word, strict_vows,
      vowel_scoring):
170     #get similar vowels from input word
171     test_similar_vowel_sequence = get_similar_vowel_sequence(test_word, strict_vows)
172     similar_word_sequences = word_sequences
173     all_word_sequences = []
174     syl_num_test_word = get_syl_num(test_word)
175     nucleuses = [get_nucleus(test_word, syl_index) for syl_index in xrange(get_syl_num(
          test_word))]
176
177     for similar_word_index, word_sequence in enumerate(similar_word_sequences):
```

```
178             reordered = get_reorder(word_sequence)
179             flattened = get_flat_list(reordered[0])
180             words_vowel_sequence = get_vowel_sequence(flattened)
181             if get_syl_num(words_vowel_sequence) == syl_num_test_word:
182                 matching_syls = []
183                 #first and last vowel match exactly (but dont distinguish between long and
                        short)
184                 first_syl_nucleus_test = get_nucleus(test_word, 0)[0]
185                 last_syl_nucleus_test = get_nucleus(test_word, -1)[0]
186
187                 try: #if for some weird reason a position is None, assign an invalid value (
                        consonant) to skip the sequence
188                     first_syl_nucleus_words = words_vowel_sequence[0][0]
189                     last_syl_nucleus_words = words_vowel_sequence[-1][0]
190                 except:
191                     last_syl_nucleus_words = 'd'
192
193                 if first_syl_nucleus_words == first_syl_nucleus_test and
                        last_syl_nucleus_words == last_syl_nucleus_test:
194                     for syl_index, vowel in enumerate(words_vowel_sequence):
195                         if vowel_scoring == "diag":
196                             if vowel is not None and vowel in test_similar_vowel_sequence[
                                    syl_index]:
197                                 matching_syls.append(flattened[syl_index])
198                         elif vowel_scoring == "mel":
199                             mel_distance = get_mel_distance(vowel, nucleuses[syl_index])
200                             if vowel is not None and mel_distance < 200:
201                                 matching_syls.append(flattened[syl_index])
202                 if len(matching_syls) == get_syl_num(flattened):
203                     all_word_sequences.append(reordered)
204
205     unique = get_unique_matching_similar_vowel_sequence(all_word_sequences)
206
207     return unique
208
209
210 def get_mora(word, syl):
211     nucleus = get_nucleus(word, syl)
212     coda = get_coda(word, syl)
213     # if coda is empty and nucleus is short then one mora
214     if coda == '' and len(nucleus) == 1:
215         return 1
216     else:
217         return 2
218
219 #get meldistance
220 def get_mel_distance(row, column):
221     #some vowels are not in the table, e.g. E:
222     if row not in meldistances[0]:
223         row = row[0]
224     if column not in meldistances[0]:
225         column = column[0]
226     c = meldistances[0].index(column)
227     for line in meldistances[0:]:
228         if line[0] == row:
229             r = meldistances[0].index(row)
230             distance = float(meldistances[c][r])
231             return distance
232
233
234 def is_diphthong(nucleus):
235     if len(nucleus) > 1:
236         if nucleus[1] in vokale:
237             return True
```

```
238             else:
239                 return False
240         else:
241             return False
242 
243     #Get normalized meldistance.
244     def get_normalized_mel_distance(vow_1, vow_2, test_word):
245         value = get_mel_distance(str(vow_1), str(vow_2))
246 
247         is_diphthong1 = is_diphthong(vow_1)
248         is_diphthong2 = is_diphthong(vow_2)
249         value_with_diphthong = 0.0
250         if (is_diphthong1 == True and is_diphthong2 == False) or (is_diphthong1 == False and
                is_diphthong2 == True):
251             if is_diphthong1 == True:
252                 value_with_diphthong = value + get_mel_distance(vow_1[1], vow_2)/2
253                 value = value_with_diphthong
254             elif is_diphthong2 == True:
255                 value_with_diphthong = value + get_mel_distance(vow_1, vow_2[1])/2
256                 value = value_with_diphthong
257         elif is_diphthong1 == True and is_diphthong2 == True and vow_1[1] != vow_2[1]:
258             value_with_diphthong = value + get_mel_distance(vow_1[1], vow_2[1])/2
259             value = value_with_diphthong
260 
261         integerated =  copy.deepcopy(meldistances)
262         integerated = integerated[1:]
263         for i in range(len(integerated)):
264             integerated[i].pop(0)
265         flatlist = get_flat_list(integerated)
266         flatlist = [int(i) for i in flatlist]
267         maximum = 200.0
268         minimum = 0.0
269         desired_range_a = 0.0
270         desired_range_b = 2.5 #* test_word_syl_num
271         return ((desired_range_b-desired_range_a)*(value-minimum))/(maximum-minimum) +
                desired_range_a
272 
273 
274     def vowel_scores(test, match, index):
275 
276         mora_test = get_mora(test, index)
277         mora_match = get_mora(match, index)
278         test_nucleus = get_nucleus(test, index)
279         match_nucleus =  get_nucleus(match, index)
280         try:
281             if test_nucleus[1] in vokale:
282                 test_nucleus_is_diphthong = True
283             else:
284                 test_nucleus_is_diphthong = False
285         except:
286             test_nucleus_is_diphthong = False
287         try:
288             if match_nucleus[1] in vokale:
289                 match_nucleus_is_diphthong = True
290             else:
291                 match_nucleus_is_diphthong = False
292         except:
293             match_nucleus_is_diphthong = False
294 
295         if (test_nucleus_is_diphthong == True and match_nucleus_is_diphthong == True) or \
296         (test_nucleus_is_diphthong == False and match_nucleus_is_diphthong == False):
297             both_diphthongs_or_not_diphthongs = True
298         else:
299             both_diphthongs_or_not_diphthongs = False
```

```
300
301         if test_nucleus == match_nucleus and mora_test == mora_match and
                both_diphthongs_or_not_diphthongs == True:
302             return 0
303         elif test_nucleus == match_nucleus and mora_test != mora_match and
                both_diphthongs_or_not_diphthongs == True:
304             return 0.5
305         elif (test_nucleus == match_nucleus and mora_test == mora_match and
                both_diphthongs_or_not_diphthongs == False) or \
306         (test_nucleus != match_nucleus and mora_test == mora_match and
                both_diphthongs_or_not_diphthongs == True):
307             return 1
308         elif (test_nucleus != match_nucleus and mora_test != mora_match and
                both_diphthongs_or_not_diphthongs == True) or \
309         (test_nucleus == match_nucleus and mora_test != mora_match and
                both_diphthongs_or_not_diphthongs == False):
310             return 1.5
311         elif test_nucleus != match_nucleus and mora_test == mora_match and
                both_diphthongs_or_not_diphthongs == False:
312             return 2
313         elif test_nucleus != match_nucleus and mora_test != mora_match and
                both_diphthongs_or_not_diphthongs == False:
314             return 2.5
315         else:
316             print 'that␣should␣not␣have␣happened'
317             print mora_test
318             print mora_match
319             print test_nucleus
320             print match_nucleus
321             print both_diphthongs_or_not_diphthongs
322
323     # add weights for vowels and sort result of get_matching_similar_vowel_sequence by
            distance
324     def sort_vowels_by_distance(matches, test_word, ignore_word_boundaries, vowel_scoring):
325         vow_scores = []
326         nucleuses = [get_nucleus(test_word, syl_index) for syl_index in xrange(get_syl_num(
                test_word))]
327
328         for word_index in xrange(len(matches)):
329             flattened = get_flat_list(matches[word_index][0])
330             weight_list = [[]]
331             for syl_index in xrange(len(flattened)):
332                 if vowel_scoring == "diag":
333                     vowel_score = vowel_scores(test_word, flattened, syl_index)
334                     weight_list[0].append(vowel_score)
335                 elif vowel_scoring == "mel":
336                     flat_nucleus = get_nucleus(flattened, syl_index)
337                     mel_distance = float("%.2f" % get_normalized_mel_distance(flat_nucleus,
                            nucleuses[syl_index], test_word))
338                     weight_list[0].append(mel_distance)
339             vow_scores.append([matches[word_index][0]] + [matches[word_index][1]] +
                    weight_list)
340         for word_index in xrange(len(vow_scores)):
341             if vowel_scoring == "diag":
342                 vow_scores[word_index].append([sum(vow_scores[word_index][2])])
343             elif vowel_scoring == "mel":
344                 vow_scores[word_index].append([float("%.2f" % sum(vow_scores[word_index][2])
                        )])
345         sorted_by_score_sum = sorted(vow_scores, key=lambda x: x[3])
346         if ignore_word_boundaries == False:
347             sorted_by_score_sum = [sorted_by_score_sum[index] for index in xrange(len(
                    sorted_by_score_sum)) if len(sorted_by_score_sum[index][0]) == 1]
348         return sorted_by_score_sum
349
```

```
350 ############## consonants ##############
351
352 # get the coda of a syllable in a cetrain position in a word
353 def get_coda(word, pos):
354     try:
355         coda = word[pos].split(get_nucleus(word, pos), 1)[1]
356         for i in coda:
357             if i in vokale or i == ':':
358                 coda = coda.replace(i, "")
359         return coda
360     except IndexError:
361         return ''
362
363 # in consonanthood.txt consonants are sorted by their consonanthood category:
364 # glides (1) > rhotic (2) > lateral (3) > nasals (4) > voiced fricatives (5)
365 # > voiceless fricatives (6) > voiced stops (7) > voiceless stops (8)
366 # get the index of consonanthood
367 def get_consonant_index(cons):
368     for i in consonanthood:
369         if i[0] == cons:
370             return int(i[1])
371
372 # get the number of categories in consonanthood.txt
373 def get_consonant_category_number():
374     temp = {entry[1] for entry in consonanthood}
375     return len(temp)
376
377 # get consonant pair weight from consonanthood.txt by taking the distance between
378 # the categories and averaging. used as substitution weight in coda_distance
379 def get_consonant_pair_weight(cons1, cons2):
380     cat_num = get_consonant_category_number()
381     cons1_index = get_consonant_index(cons1)
382     cons2_index = get_consonant_index(cons2)
383     return abs((cons2_index - cons1_index)/float(cat_num-1))
384
385
386 #levenshtein modified to add a weight on substitution, depending on consonant similarity
387 def coda_distance(s1, s2):
388     if len(s1) < len(s2):
389         return coda_distance(s2, s1)
390
391     # len(s1) >= len(s2)
392     if len(s2) == 0:
393         return len(s1)
394
395     previous_row = range(len(s2) + 1)
396     for i, c1 in enumerate(s1):
397         current_row = [i + 1]
398         for j, c2 in enumerate(s2):
399             insertions = previous_row[j + 1] + 1 # j+1 instead of j since previous_row
                  and current_row are one character longer
400             deletions = current_row[j] + 1       # than s2
401             if c1==c2:
402                 subst_weight = 0
403             else:
404                 try:
405                     if c1 in r':6' or c2 in r':6':
406                         subst_weight = 1
407                     else:
408                         subst_weight = get_consonant_pair_weight(c1, c2) + 1
409                 #e.g. if vowels of a diphtong are found: in ae a is considered the
                        nucleus, e is left in coda. Just ignore the vowel in this case.
410                 #also ignores '"', ~ and such.
411                 except:
```

```
412                     subst_weight = 0
413                 substitutions = previous_row[j] + subst_weight
414                 current_row.append(min(insertions, deletions, substitutions))
415             previous_row = current_row
416         return previous_row[-1]
417 
418     # adds consonant weigts to the vowel list and sorts by scores of vowels and consonants
419     def sort_consonants_by_distance(sorted_vowels, test_word):
420         test_codas = []
421         for syl_index in range(len(test_word)):
422             test_codas.append(get_coda(test_word, syl_index))
423         for word_index in range(len(sorted_vowels)):
424             cons_weights = []
425             word_codas = []
426             word_sequence = get_flat_list(sorted_vowels[word_index][0])
427             #print word_sequence
428             for syl_index in range(len(word_sequence)):
429                 word_codas.append(get_coda(word_sequence,syl_index))
430             for coda_index in range(len(test_codas)):
431                 cons_weights.append(float("%.2f" % coda_distance(test_codas[coda_index],
                        word_codas[coda_index])))
432             sorted_consonants=sorted_vowels
433             sorted_consonants[word_index].append(cons_weights)
434             sorted_consonants[word_index].append([float("%.2f" % sum(cons_weights))])
435             sorted_consonants[word_index].append(float("%.2f" % sum(sorted_consonants[
                    word_index][2]+sorted_consonants[word_index][4])))
436         if len(sorted_vowels) == 0:
437             return []
438         sorted_by_score_vow_and_cons = sorted(sorted_consonants, key=lambda x: x[6])
439         return sorted_by_score_vow_and_cons
440 
441     ############## others ##############
442 
443     # remove postags
444     def remove_postag(words):
445         pos_removed = [words[word].split(",", 1)[0] for word in range(len(words))]
446         cleartext=[words[word].split(",")[2].strip('\n') for word in range(len(words))]
447         joined = [i for x, y in zip(pos_removed,cleartext) for i in [[x, [y]]]]
448         return joined
449 
450     # bring everything together to get a list of rhymes
451     # vowel_scoring either done with vowel diagram (diag) or with mel distances (mel)
452     def final(words, test, vows, vowel_scoring, ignore_word_boundaries = True, strict_vows =
            False):
453         print "get word sequences"
454         word_sequences = get_word_sequences(words, test)
455         print "get similar vowel sequence"
456         matching_similar_vowel_sequence = get_matching_similar_vowel_sequence(word_sequences
                , test, strict_vows, vowel_scoring)
457         sorted_vows = sort_vowels_by_distance(matching_similar_vowel_sequence, test,
                ignore_word_boundaries, vowel_scoring)
458         print "sort by consonants"
459         sorted_cons = sort_consonants_by_distance(sorted_vows, test)
460         return sorted_cons
461 
462     if __name__ == '__main__':
463 
464         test_words = strip_list_test(test_words)
465         query = test_words[7]
466 
467         print query
468         print "reading corpus"
469         words = strip_list(remove_postag(readFiles()))
470         print "words in corpus: ", len(words)
```

```
    vows = readVowelMatrix()
    consonanthood = readConsonantHood()

    sorted_cons = final(words, query, vows, "diag", True, False)

    for i in sorted_cons[0:20]:
        print flatten(i[1]), i[0], i[2], i[3], i[4], i[5], i[6]

    print "results: ", len(sorted_cons)
```