Norwegian poetry generation and rhyme modelling

Tita Enstad

Thesis submitted for the degree of
Master in Informatics: Language Technology
60 credits

Institute for Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Norwegian poetry generation and rhyme modelling

Tita Enstad

Abstract

Computational Creativity is by some seen as the ultimate challenge for AI. Numerous works on the computationally creative task of poetry generation have been published. However, no contributions on this topic have been made for Norwegian. In this thesis, we address that gap.

We present NoRSC: Norwegian Rhyme Scheme Corpus, a publicly available rhyme scheme annotated data set of Norwegian poetry. Using the NoRSC data set, we train LSTM-based models on rhyme and poetry generation.

We explore combining rhyme models and language models to enhance rhyming ability in poetry generation. Our poetry generation model can generate a stanza with any rhyme scheme.

Our poetry generation model is evaluated with human evaluation. We investigate whether people believe that the generated poetry is written by a human in a Turing-like test. The evaluators are also asked to score the rhyme in the generated stanzas.

We received 27 answers to our web forms, and could conclude that our best poetry generation model achieved high rhyme scores, and was in about half of the instanced able to imitate a human poet.

Acknowledgements

I would like to thank Lilja Øvrelid for being my supervisor these past two years. I left every biweekly supervision meeting feeling more motivated than I was before going in.

I would also like to thank Språkbanken (*eng: The Norwegian Language Bank*) for being so helpful with sharing their data, even when I ended up not using all of it in the end. A special thanks to Magnus who helped me with several rounds of re-OCR of their older material.

I am also very grateful to Språkrådet (*eng: The Language Council of Norway*), who awarded me their master's degree stipend[1]. It has helped me to be able to focus on my studies full time.

A special thanks to Petter Mæhlum, who helped me with the annotation of my data set, as well as sharing his knowledge on measuring inter annotator agreement.

Last of all I would like to thank Markus Sverdvik Heiervang, who has helped me both with academics and motivation. Thank you for always being by my side. I would not have been able to finish this work without your continuous support.

[1] https://www.sprakradet.no/Vi-og-vart/Stipend/mastergradsstipend-sprak-og-ikt-2021/

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In this thesis, we explore the task of *poetry generation*, a task in the intersection of computational creativity and natural language processing (NLP). Poetry generation is, as the name suggests, the task of using computers to generate poetry. It is usually a goal to create poetry that can pass as poetry written by a human. This is closely related to the Turing test; if a person believes it is communicating with another person, when it is in fact communicating with a computer, then the computer has passed the test (Saygin, Cicekli, and Akman, 2000).

The task of automatically generating poetry has a history that dates back to the 1960s[1], and is still researched today. Several papers on poetry generation have been published in the past five years. The dominating languages are Chinese (Yi et al., 2018; Yang et al., 2018; Y. Liu, D. Liu, and Lv, 2020) and English (Lau et al., 2018; Jhamtani et al., 2019; Van de Cruys, 2020), with occasional papers on poetry generation for other languages, for example Finnish (Hämäläinen and Alnajjar, 2019), Italian (Zugarini, Melacci, and Maggini, 2019) or Russian (Tikhonov and Yamshchikov, 2018).

Another task we explore is rhyme modelling. Rhyme as a literary tool is ubiquitous in traditional poetry, song lyrics and rap lyrics (Berge, 2016). Rhyme modelling has been done both as a part of works on poetry and lyric generation (Lau et al., 2018; Nikolov et al., 2020; Van de Cruys, 2020), and independently (Haider and Kuhn, 2018).

1.1 Objectives and research questions

The main objective of this thesis is to address obvious gaps in the fields of poetry generation and rhyme modelling: no publicly available work has been done for Norwegian (to the best of our knowledge).

Our modelling objective is to investigate whether we can create a Norwegian poetry generation model that produces poetry of a quality such that

1. it cannot be discerned from poetry written by humans and
2. it rhymes.

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

In order to train such a model, we need rhyme scheme annotated training data. Thus, our second objective is to collect and annotate a data set of Norwegian rhyming poetry. A significant part of the work on this thesis is the creation of NoRSC: Norwegian Rhyme Scheme Corpus.

In order to evaluate our poetry generation model, we use human evaluation. We want to find out if the generated poetry is perceived to be written by a human (a Turing-like test) and how well it is perceived to rhyme.

1.1.1 Research questions

Our research questions are listed below.

RQ1: Is there a strong enough relation between written Norwegian and spoken Norwegian to accurately model rhyme based on text data?

RQ2: How consistent are our poetry generation models at generating rhyming poetry?

RQ3: To what degree is our generated poetry believed to be written by a human?

RQ4: Is there a connection between the generated poetry rhyming and it being perceived to be written by a human ?

To specify **RQ1**: for our rhyme modelling, we only use the rhyme relations extracted from the annotated data set, and no other phonetic information. Thus, the rhyme models only have access to the words themselves, and the character sequences that form them. Whether two words rhyme is dependent on the phonology of the words. To successfully model rhyme based on the character sequences that make up the words, there has to be a strong relation between how the words are written and how they are pronounced.

We aim to answer **RQ2**, **RQ3** and **RQ4** by analyzing the results of our human evaluation.

1.2 Thesis outline

In chapter 2 we discuss the poetry genre, rhyme and poetry generation as an NLP task. We present some recent work that is relevant to our modelling objectives, describing the data sets, the model architecture and the evaluation methods.

In chapter 3, we present NoRSC: Norwegian Rhyme Scheme Corpus, a data set of rhyme scheme annotated Norwegian poetry, as well as the work that went into creating the data set.

In chapter 4, we describe how we extract rhyme pairs from the NoRSC data set, and how we can harvest additional pairs with a graph based approach.

In chapter 5, we use the rhyme pairs from the previous chapter for rhyme modelling. We train LSTM-based models for two different rhyme modelling tasks: rhyme detection and rhyme generation.

In chapter 6, we design and train LSTM-based poetry generation models on the NoRSC data set. We evaluate the produced poetry with human evaluation, where both a Turing-like test and rhyme ratings are performed.

In chapter 7, we summarize our findings, finally address the research questions and conclude the thesis.

,

Chapter 2

Background

In this chapter, we discuss rhyme and poetry, poetry generation and rhyme modelling. We explore recent work that is relevant to the topic of our research, with a focus on the model architecture and the evaluation methods. Here, we lay the foundation for our work in the coming chapters.

2.1 Rhyme and poetry

Poetry can be divided into fixed form and open form poetry. Fixed form poetry has rules for structure, such as the number of lines, number of syllables for each line, stress and/or rhyme. Examples include haikus, sonnets and limericks. With open form poetry, there are no rules. While open form poetry is most common today, it is typically in fixed form poetry we find rhyme (Berge, 2016).

2.1.1 Rhyme schemes

Some types of poems have a specified *rhyme scheme* they have to follow. A rhyme scheme is the pattern of rhyme in a poem. It is typically annotated with letters in alphabetical order. For example, the limerick is a five-line poem with an AABBA rhyme scheme (Jhamtani et al., 2019). See 2.1 for an example of a rhyme annotated limerick.

(2.1)	There was an Old Man with a beard,	A
	Who said, "It is just as I feared!—	A
	Two Owls and a Hen,	B
	Four Larks and a Wren,	B
	Have all built their nests in my beard!"	A
	(Lear, 2008)	

All lines annotated with the same letter rhyme with each other. The first, second and fifth lines are annotated with A, and the third and fourth with B. We see that the *line-ending words* (the last word in a line) for the lines annotated with A rhyme: beard and feared. The same is true for the line-ending words in the lines annotated with B; hen and wren. Though not all poetry have strict rhyme scheme rules, all poetry can be annotated with its rhyme scheme, which is crucial for supervised learning.

2.1.2 Rhyme pairs

From rhyme scheme annotated data, we can extract *rhyme pairs*. Rhyme pairs are pairs of words or word sequences that rhyme. An example of a rhyme pair is ("cat", "hat").

Similarly, negative rhyme pairs are pairs of words or word sequences that do not rhyme. An example of a negative rhyme pair is ("shoelaces", "stubborn").

While the rhyme is often contained in singular words, there are examples where the rhyme spans several words. In Example 2.2 below, the rhyme pair is ("sent it", "meant it"). We call this *multiword rhyme*.

(2.2) I can't believe he sent it,
I didn't think he meant it.

2.2 Poetry generation

Poetry generation is a specific type of the more general task natural language generation (NLG).

2.2.1 Natural language generation

Natural language generation is the task of using computers to generate (what appears to be) natural language. There is a vast number of sub-tasks, such as translation, summarization, question answering and robo-journalism (Gatt and Krahmer, 2018).

The field of NLP in general has had an influx of new techniques for language modelling in recent years, which has led to dramatic improvement with regards to performance. Vaswani et al.'s famous "Attention is all you need" from 2017 introduced the transformer architecture, which revolutionized the field (Gillioz et al., 2020). This gave way to central contributions in the field like BERT (Devlin et al., 2019) and GPT (Radford, Narasimhan, et al., 2018; Radford, J. Wu, et al., 2019; Brown et al., 2020). Transformer based models such as T5 (Raffel et al., 2020) and GPT-3 (Brown et al., 2020) currently dominate NLG, pushing forward state-of-the-art on multiple tasks.

2.2.2 Evaluating poetry generation systems

For poetry generation, human evaluation is the dominant methodology. The model is supposed to *create* something *new*, so a gold data set simply cannot exist. Artistic taste is highly subjective, which makes it difficult to reproduce experiments where the human judges rate how *good* the generated poetry is.

As mentioned in chapter 1, a Turing-like test is usually performed, to see if the generated poetry is believed to be written by a human or a computer. Multiple previous works conduct human evaluation by presenting a generated and human-written poem side by side, and ask the evaluator to pick out the human-written/generated poem (Lau et al., 2018; Jhamtani et al., 2019). Other works evaluate by presenting the poems individually (Van de Cruys, 2020) and some do both (Nikolov et al., 2020).

2.2.3 “The great misalignment” in human evaluation for NLP

Hämäläinen and Alnajjar (2021) discuss what they call the “Great Misalignment Problem” in human evaluation of NLP methods. They found that out of 10 randomly selected papers published in the ACL 2020[1] that use human evaluation, only 1 of the papers evaluated the model in line with the definition of the problem the model was supposed to solve.

They conclude that “[in the field of NLP] human evaluation is not conducted in the same rigorous fashion as in other fields dealing with human questionnaires such as in social sciences or fields dealing with evaluation of computer systems such as design science” and that “There is a long way for our field to go from here in order to establish more sound and reproducible human evaluation practices.”

2.2.4 Intrinsic evaluation

For models that generate poetry based on an input text, evaluation metrics for machine translation such as BLEU (Papineni et al., 2002) or other ways to measure content overlap can be used (Nikolov et al., 2020). Human evaluation is still the preferred evaluation for such task, but intrinsic evaluation can still give insights, and play a role in model selection.

2.3 Rhyme modelling

Rhyme modelling refers to a number of modelling tasks where the objective is understand rhyme. Such tasks include:

- rhyme detection: the objective is to predict whether two words rhyme
- rhyme generation: the objective is to generate one or more words that rhymes with the input word
- rhyme scheme identification: the objective is to tag a stanza with its rhyme scheme

Training data for rhyme models is typically rhyme scheme annotated text or a collection of rhyme pairs. While there are papers on rhyme modelling independent of any NLG system (Haider and Kuhn, 2018; Addanki and D. Wu, 2013), rhyme modelling is often used as an auxiliary task for NLG systems to enhance rhyming ability in generation of rhyming text. For example, Lau et al. (2018) train a rhyme detection model to sample line-ending words as a part of their larger poetry generation model.

2.3.1 Evaluating rhyme models

Rhyme models can be evaluated by using pronouncing dictionaries. To check if two words rhyme, one can look up the phonetic representation, and check for overlap in the last syllable(s) of both words.

[1]https://www.aclweb.org/anthology/events/acl-2020/

For English, there are several public pronouncing dictionaries, such as The CMU pronouncing dictionary[2] (over 134,000 words). The most common words in the English version of Wiktionary have a “Pronunciation”-section, where the word is transcribed with the International Phonetic Alphabet (IPA).

There also exists a pronouncing dictionary for Norwegian, the NLB Pronunciation Lexicon for Norwegian Bokmål[3]. However, we do not explore the application of the NLB Pronunciation Lexicon in this thesis.

2.4 Relevant previous works

According to Lau et al. (2018), Greene, Bodrumlu, and Knight (2010) was the earliest attempt at using statistical modelling for poetry generation. Before that, poetry generation methods were rule-based, using syllable counting and rhyme dictionaries (Lau et al., 2018). As explained in section 2.2.1, the field of NLG has changed dramatically in the past few years. Thus, one could expect the field of poetry generation to see a similar development, and that recent models perform much better than traditional, rule-based models. Because of this, we will focus on research published within the past 5 years. We will also limit our focus to work that describe models that generate texts that rhyme, as this is in line with our research objectives.

2.4.1 “Deep-speare: A joint neural model of poetic language, meter and rhyme”

Lau et al. (2018) present DEEP-SPEARE; a language model that generates Shakespearean sonnets. A Shakespearean sonnet (from now on, sonnet) is a special type of poem that has rules for stress, structure and rhyme. The model is comprised of three sub-models: a language model, a stress model and a rhyme model. All sub-models use variants of LSTMs, and they are trained together, allowing them to influence each other during training.

The rhyme model is a unidirectional LSTM that learns to separate rhyming and non-rhyming word pairs. Making use of the fact that a quatrain[4] contains rhyming pairs, we can sample both positive and negative rhyming pairs by extracting the last word of each line in a quatrain. For each target word t, there will be one positive rhyme pair (t, x) and two negative rhyme pairs (t, y), (t, z). Adding additional negative rhyme pairs by sampling random words from the vocabulary increased performance.

The set of rhyme pairs for a target word are one instance of training. The character embeddings for all words are extracted from the shared character embedder as used by other sub-models. These character embeddings are fed to the LSTM, outputting word representations u_t, u_x etc. The loss function is a margin based loss calculated by using the cosine similarity of the outputted character-based word representations. The LSTM will thus be trained to output similar vectors for words that rhyme.

[2]https://github.com/cmusphinx/cmudict

[3]https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-52/

[4]a stanza of four lines in a poem

Training data

The model was trained on an English sonnet corpus extracted from Project Gutenberg, resulting in 3355 sonnets and 367 000 words. The number of unique words is not stated, as far as we can see.

Evaluation and results

Each sub-model is evaluated separately on gold data. The rhyme model is evaluated by extracting phonetic information from the CMU pronouncing dictionary, and comparing produced word pairs. It seems that the model predicts a high cosine similarity for words that have the same sub-sequences of characters. This produces false positives such as predicting rhyme for (overgrown, frown) and (afraid, said), and false negatives such as not predicting rhyme for (never, endeavour).

The sonnet generator itself is evaluated by humans, both by crowd workers and expert evaluation. The crowd workers are presented with two poems, one real and one computer generated, and are asked to point out the computer generated one. The crowd workers' accuracy on identifying the computer generated poem became as low as 53% for the best model.

The expert is asked to rate poems on 4 aspects: meter, rhyme, readability and emotion, on a scale from 1 to 5, where 5 is the best score. The model received better scores for rhyme and meter than human poems, though this might not be *that* surprising, as human artists tend to break the rules on purpose. The author sums it up well:

> Despite excellent form, the output of our model can easily be distinguished from human written poetry due to its lower emotional impact and readability. In particular, there is evidence here that our focus on form actually hurts the readability of the resulting poems, relative even to the simpler language models.
>
> (Lau et al., 2018)

2.4.2 "Learning Rhyming Constraints using Structured Adversaries"

Jhamtani et al. (2019) present RHYME-GAN, a Generative Adversarial Network (GAN) model that learns the rhyming constraints directly from the training data. The paper aims to combat some of the drawbacks of Lau et al. (2018), such as the fact that all sonnets are generated by rejection sampling[5], and that the training data must be quatrains where each line-ending word must rhyme with one and only one of the other line-ending words.

As all GANs, the model consists of a generator and a discriminator, where the discriminator is trained to distinguish between real and generated poems, and the generator is trained to "fool" the discriminator.

The generator generates poems by first generating line-ending words, and then generating the lines backwards conditioned on the line ending words. The discriminator takes as input a poem, and outputs the probability that this

[5]Rejection sampling means to continue to sample values from a probability distribution (in this case, a language model) until the sampled value is acceptable.

poem is real, i.e. part of the training data, and not generated. On input a poem of T lines, the discriminator creates a similarity matrix $S = R^{T \times T}$, to capture the similarity of the line-ending words in the poem. The line-ending words are encoded with a character-based LSTM, and the similarity is calculated by computing the cosine similarity of these character based word embeddings. Finally, the output signal of the discriminator is a 2D convolutional layer applied to the similarity matrix S, passed through the sigmoid function.

Training data

The model was trained on two different corpora, the sonnet corpus from Lau et al. (2018), and a limerick corpus obtained by web scraping. The limerick corpus contains a train set of 10 400 limericks, and 1300 limericks in the validation and test sets.

Evaluation and results

Since both DEEP-SPEARE and RHYME-GAN measure rhyme by character sequence similarity, one can expect similar problems as mentioned in 2.4.1, though this is not explicitly mentioned in Jhamtani et al. (2019).

The model is evaluated the same way as the non-expert human evaluation in Lau et al. (2018), that is, human judges are asked to correctly identify the computer generated poem when shown one "real" and one generated poem. Evaluation was done on 150 valid generated poems, where valid means that the poem has the correct rhyme scheme for its' type (limerick: AABBA, sonnet: ABAB or ABBA or AABB). Determining the validity of the poems was done by extracting phonetic information from the CMU pronuncing dictionary.

For the 150 examples, the human judges had an accuracy of 56%, which is slightly higher than Lau et al. (2018)'s 53%. Remember that lower accuracy means that it is more difficult for the human judges to distinguish between poems written by humans and computer generated ones. However, using pre-rejection samples from both RHYME-GAN and DEEP-SPEARE, the annotators achieve accuracies of 60% and 81%, showing that RHYME-GAN produces believable poetry at a much higher frequency than DEEP-SPEARE.

2.4.3 "Automatic Poetry Generation from Prosaic Text"

Van de Cruys (2020) present a model that generates rhyming poems, though it's only trained on "normal" prosaic text. The core of the system is a model, that given an input sentence S_i, predicts the next sentence S_{i+1}. It uses an encoder-decoder architecture with gated recurrent units (GRU), combined with an attention mechanism.

Each word of the input sentence S_i is sent through the decoder and the hidden state is computed. This hidden state is in turn sent to the decoder, where the next sentence S_{i+1} is predicted. The decoder will produce S_{i+1} in reverse order, so the last word of S_{i+1} is predicted first, enabling end rhyme constraints.

For the rhyme constraints, Van de Cruys (2020) use the phonetic representations of words as given by Wiktionary. For every word in the vocabulary of the model, a set of rhyming words are extracted. In the decoder-encoder architecture, a prior probability distribution p_{rhyme} is included the calculations, ensuring that the predicted sentence will rhyme. Both encoder and decoder consist of a double GRU layer with 2048 hidden nodes. When producing poems, the model would produce 2000 candidate verses, and choose the best one based on several criteria. They used a fixed rhyme scheme for all poems: ABAB CDCD.

Training data

Two models were trained, one for English and one for French. Both were trained on a 500 million word corpus, with a vocabulary of 15000 words.

Evaluation and results

Van de Cruys (2020) use the evaluation framework presented by Zhang and Lapata (2014), who suggest these four aspects for human evaluation of computer generated poetry. Each poem is rated from 1-5 on each aspect, where 5 is best, and 1 is worst.

- fluency: is the poem grammatical and syntactically well-formed?
- coherence: is the poem thematically structured?
- meaningfulness: does the poem convey a meaningful message to the reader?
- poeticness: does the text display the features of a poem?

The generated poems achieved good scores, above 3/5 across the all the four aspects. The scores were somewhat higher for the French model than the English one, but on the question of whether the poem was written by a human, the English model scored better. For the English model, 59% believed the poems to be written by a human, but for the French one it was only 45%.

2.4.4 "Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders"

Nikolov et al. (2020) present the RAPFORMER, a transformer-based denoising autoencoder that produces rhyming rap lyrics. The RAPFORMER takes in any input text, and will output the same content, but as rap lyrics.

These three steps make up the models algorithm:

1. The content words are extracted from each line of the input text
2. A transformer model builds a rap verse from the content words
3. Line-ending words are substituted with suitable words to create rhyme

For step 1, in addition to extract the context words from the original input text, the words are shuffled, 20% of the words are dropped, and 20% are replaced by synonyms from WordNet Miller (1995). This noise is added to create variation.

The model in step 2 is a 6-layer transformer encoder-decoder trained on a corpus of rap lyrics, where it is provided with the content words as described in step 1, and is trained to reproduce the original verse. This lets the model learn the latent structures of rap lyrics, given content words.

Step 3 involves using a BERT-base model that has been fine-tuned on the rap corpus to predict the 200 most likely substitutions for the line-ending word. Out of these 200, the one that increases rhyme the most is chosen, using the same notion of "rhyme" as Malmi et al. (2015): the longest overlapping of vowels between the chosen word and the word it is supposed to rhyme with. This means that the rhyme may span several words, but also that it accepts words that do not technically rhyme. The rapper may have more freedom to twist the way a word is pronounced to make it sound more like a rhyme, which can explain this choice.

Training data

They use three different data sets to ensure that the model works well for different types of input. The rap data set consist of 60,000 song lyrics from the lyric website musixmatch[6], reserving 2,000 songs for development and testing. The out-of-domain data sets are the news summaries data set from Hermann et al. (2015) and a subset of the movie plot summary data set from Bamman, O'Connor, and Smith (2013). The 50,000 most frequent words in the joint data sets are used as the vocabulary for the model.

Evaluation and results

The outputted lyrics are measured on rhyme density (RD) (Malmi et al., 2015) and content word overlap between input and output texts, as well as BLEU (Papineni et al., 2002) when the model is trained on rap reconstruction.

They also use human evaluation, with three judges with domain knowledge. For the rap lyrics generated from news text input, the human judges were presented with 100 generated lyrics, and asked three questions:

1. How much do the lyrics presented resemble rap lyrics? (1 to 5)
2. How well do the lyrics preserve the content of the original news article? (1 of 5)
3. Do these lyrics look like a song you know? (yes/no)

The RAPFORMER didn't perform that well, with a mean score of 2.03 for 1, 2.55 for 2 and 8% yes answers for 3.

They also perform two Turing-test-inspired experiments with 100 generated and original lyrics:

1. The original lyrics and the RAPFORMER lyric are presented side by side, and the human judge is asked to pick the lyric written by a human.
2. The human judge is presented with a lyric and asked to determine if it is AI-generated or written by a human.

[6]https://www.musixmatch.com/

For 1, only 7% of the texts are mislabelled. For 2, 25% of the texts are mislabelled, which means that the human judges were able to tell real lyrics apart from generated lyrics 75% of the time.

2.5 Earlier works on rhyme detection

These are not works on poetry generation, but rather rhyme detection. Both represent words as character vectors and train siamese LSTM-networks to predict rhyme between two sequences. These kinds of experiments are good indications of how well written text can represent the phonetics of rhyme in the respective languages.

2.5.1 "Supervised Rhyme Detection with Siamese Recurrent Networks"

This paper from 2018 presents another method for rhyme detection than covered in the papers in section 2.4. In this paper, Haider and Kuhn train Siamese Recurrent Networks (SRN) to predict if two sequences rhyme. They describe their model architecture as three layers of character-based BiLSTMs with 50 hidden units, followed by a dense feed-forward layer. If the model outputs >0.5, the two input sequences are predicted to rhyme.

They use 5000 German rhyme pairs, and generate negative rhyme pairs by shuffling the positive. The character embeddings have a dimension of 100, and they train for 100 epochs. They train the SRN on positive and negative rhyme pairs with a 2:3 ratio, and end up with an accuracy of 96%. Similarly, they train an English version, this time with 10,000 and 30,000 rhyme pairs and end up with accuracies of 96% and 97%, respectively. See Haider and Kuhn (2018) for specific details.

2.5.2 "Using Siamese neural networks to create a simple rhyme detection system"

Minogue (2021) collected rhyme annotated rap lyrics in English from the song lyric website Genius, and trained a SRN to predict rhyme. They used an equal number of positive and negative rhyme pairs, 500 000 each, or 1 000 000 in total. The data was split into train/validation/test sets with the train set being 60% of the data, the test set 30% and the validation set 10%. The model architecture is one siamese LSTM layer with 64 hidden units, followed by three dense layers of decreasing size before the output layer. Minogue end up with an accuracy of 95% on the test set (Minogue, 2021).

2.5.3 Comparison

Minogue (2021) and Haider and Kuhn (2018) receive similar results, with the difference in accuracy being just 1-2 percentage points. While Haider and Kuhn (2018) use a lot less data, they have a more complex architecture, with three bi-LSTM layers. Minogue (2021) on the other hand, has a simpler architecture, but 100 times more data than Haider and Kuhn (2018).

2.6 Summary

We have seen that many different types of NLP architectures can be used for poetry generation. Mentioned above are several different types of RNNs (LSTM, BiLSTM, GRU), transformer-based models, GANs and even reinforcement learning. All articles used some form of human evaluation for their generated texts, but as they a) did not use the same questions during evaluation and b) had a different number of examples, different number judges, and different level of domain expertise for the judges, it is hard to say that one is better than the others.

We also discussed different ways to model rhyme. Haider and Kuhn (2018) and Minogue (2021) train Siamese LSTMs for rhyme detection. Some works use the similarity of character sequence to model rhyme (Lau et al., 2018; Jhamtani et al., 2019), but evaluate on pronunciation dictionaries, while other use the pronunciation dictionary directly during generation (Van de Cruys, 2020; Zugarini, Pasqualini, et al., 2021). In Nikolov et al. (2020), they measure rhyme as the longest overlap of vowels, a less strict definition of rhyme than other papers.

Chapter 3

Creating a corpus of rhyme scheme annotated Norwegian poetry

One goal of this thesis is to create a freely available rhyme scheme annotated corpus of Norwegian poetry. We call it NoRSC: Norwegian Rhyme Scheme Corpus. In this chapter we describe the source corpus, the pre-processing pipeline, the annotation process and rhyme scheme statistics.

The data set consists of texts that were already in the public domain, so it is available for anyone who might want to use it here[1].

3.1 Source corpus

The poems for the data set were extracted from the "Public Domain Texts from NBdigital" corpus[2] from Språkbanken, The Norwegian Language Bank at The National Library of Norway. According to the corpus website, the corpus is made up of 26.344 books in .txt and .xml formats, that were produced by running optical character recognition (OCR) on scans of physical books. All of these texts are in the public domain, which means that unless the authors willingly renounced their copyright of the text, at least 70 years have passed since their death (as per Norwegian copyright laws (Åndsverkloven, 2018)).

3.2 Extracting the poetry

We downloaded and unpacked the entire source corpus, which consisted of 21.483 files total (for some reason not 26.344 as stated on the corpus website). All files are named on the same format, with the digibok identifier, year of publication, language, author name and book title separated by dashes (see example 3.1 below). The book title is cut at 45 characters if it is any longer.

[1]https://github.com/titaenstad/norwegian_rhyme_scheme_corpus

[2]https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-34/

(3.1) `digibok_2009013000029-1894-nob-728-Bugge_Sophus-Bidrag_til _den_eldste_Skaldedigtnings_Histori.txt`

Using a set of heuristics implemented as regular expressions, we extracted all the books that we believed to contain rhyming texts, based on the file name. Not only purely poetry, but also songs and psalms were included in our wider definition of poetry. This is because we wanted to capture as many rhyming texts as possible. Thus, we extracted all books where the file name matched using the simple regex seen in example 3.2. The regex matches with strings containing the words *dikt* 'poem' written with both k and g, *lyrikk* 'poetry' written with both single and double consonant k, *salme* 'psalm' and *sang* 'song'. The variations of the words *dikt* and *lyrikk* are to include books where older spelling is used.

(3.2) `.*[dD]i[g|k]t.*|.*[lL]yrik.*|.*[Ss]alme.*|.*[Ss]ang.*`

This filtering stage resulted in a list of 593 book titles which we were to use to build our poetry corpus. One caveat of our extraction method is that it will produce some false positives: since this extraction is based on the filename, and not only book title, this can result in also extracting books where the author name matches with the regular expression. In our case, a book by geologist Ottar Jøsang was included in the extracted set, because his surname contains the substring *sang*. Another fault of this extraction method is the possibility of false negatives: books where the title is longer than 45 characters, and the latter part of the book title contains any of the wanted words from the regular expression, are not extracted. Below we describe a refinement of our filtering using metadata.

3.2.1 Using The National Library of Norway's search API

After the initial filtering described above we learnt of an API[3] that can be used to extract metadata about the books based on the digibok identifier. Using this, we could extract for example the full book title (if longer that 45 characters) and year of publication (when this was not included in the filename). However, the information from the API did not always agree with the information in the filenames.

We used the API to look up the metadata of the 593 books we had extracted for our data set to to both a) retrieve the full book title and b) subsequently match the regex to the book title. This resulted in us narrowing down the number of books to 587.

After some debugging we also realised that the digibok identifier stored in the database and in the filenames is not always the same: When fetching metadata on a book title based on the identifier from the filename, the metadata returned could contain a different digibok identifier, though the rest of the metadata such as title and author matches. For our set it is different for 38 of the 593 books.

We further wanted to exclude all books that were not in Norwegian. The filenames contain a three-letter code[4] that specify the language the book was written in. This information is also stored in the database that we access with

[3]`https://api.nb.no/`, or more specifically `https://api.nb.no/catalog/v1/items/URN:NBN:no-nb_digibok_<code>`

[4]The codes are on ISO 639-3 format, see `https://en.wikipedia.org/wiki/ISO_639-3`

Sel. #	Selection procedure	# of books
1	Regex on full filename	593
2	Regex on book title (from selection 1)	587
3	Norwegian according to filename (from sel. 2)	451
4	Norwegian according to API metadata (from sel. 3)	405
5	Norwegian according to API metadata (from sel. 2)	414

Table 3.1: Number of books selected based on different selection criteria

the API. We wanted to keep all books that have either "nor", "nob" or "nno", because these are the language codes for the various Norwegian varieties.[5]

However, when we selected books based on the language codes in the filenames, we got a different selection that when we selected based on the metadata information from the API. Selecting purely on filename returns a list of 451 books that are supposedly in Norwegian. But according to the API, 46 of these books are actually not in Norwegian, but Danish (36) or multi-language (10). If we remove these, we end up with 405 books. If we select based on the API from the beginning, we get 414 books, meaning that 9 of the books that are not Norwegian according to the filenames, are actually Norwegian according to the API. According to the filenames, 1 is multi-language and 8 are "undefined". See table 3.1 for the different results.

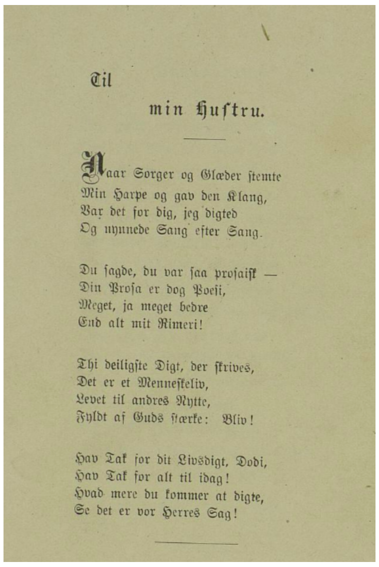
We assume that the information in the API is more recent, based on the fact that 36 books were changed from Norwegian to Danish (see more on this in section 3.4). Because of this, and because we want to keep the files that are labelled "undefined", as the book titles suggest that they are indeed Norwegian, we decided to go with the selection yielded by the API (selection 5 in table 3.1).[6]

3.3 Data set quality and OCR misreadings

A manual inspection of the extracted data makes it apparent that the texts are of pretty bad quality. Due to OCR misreadings there is a lot of noise, and sometimes it is impossible to understand what the original words are supposed to be. See example 3.3 for an example of bad quality text.

[5]"nor" refers to Norwegian, "nob" to Norwegian bokmål and "nno" to Norwegian nynorsk. The difference between these is explained later in the chapter.

[6]We acknowledge that if there are books in the original data set where the book title is longer than 45 characters, and the latter part of the book title contains any of the wanted words from the regex, these are not extracted. We did not deem it necessary to make 21 000 API calls to extract more books at the point of this discovery. This was after the annotation process had started, and we realised that we were not going to be able to annotate more than a small fraction of the extracted books.



Til

min Huſtru.

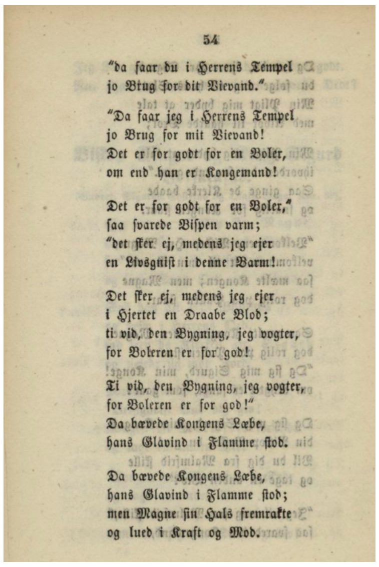
Naar Sorger og Glæder ſtemte
Min Harpe og gav den Klang,
Var det for dig, jeg digted
Og nynnede Sang efter Sang.

Du ſagde, du var ſaa proſaiſk —
Din Proſa er dog Poeſi,
Meget, ja meget bedre
End alt mit Rimeri!

Thi deiligſte Digt, der ſkrives,
Det er et Menneſkeliv,
Levet til andres Nytte,
Fyldt af Guds ſtærke: Bliv!

Hav Tak for dit Livsdigt, Dodi,
Hav Tak for alt til idag!
Hvad mere du kommer at digte,
Se det er vor Herres Sag!

Figure 3.1: Page scan from the source corpus



54

"da faar du i Herrens Tempel
jo Brug for dit Vievand."

"Da faar jeg i Herrens Tempel
jo Brug for mit Vievand!
Det er for godt for en Boler,
om end han er Kongemand!

Det er for godt for en Boler,"
ſaa ſvarede Biſpen varm;
"det ſker ej, medens jeg ejer
en Livsgniſt i denne Barm!

Det ſker ej, medens jeg ejer
i Hjertet en Draabe Blod;
ti vid, den Bygning, jeg vogter,
for Boleren er for god!

Ti vid, den Bygning, jeg vogter,
for Boleren er for god!"
Da bævede Kongens Læbe,
hans Glavind i Flamme ſtod.

Da bævede Kongens Læbe,
hans Glavind i Flamme ſtod;
men Magne ſin Hals fremrakte
og lued i Kraft og Mod.

Figure 3.2: Page scan from the source corpus

(3.3) m
min Jfjuftru.
Sorger og ©tøber [temte mn $axpe og gau bon Ælang, $ar bct for btg, jegbigteb Og utjnnebe ©ang' efter ©ang.

The only real words that can be recognized are *min* 'mine', *Sorger* 'sorrows', *og* 'and', *for* 'for' and *efter* 'after'. The words also contain characters that generally do not belong in Norwegian words, such as "©", "[" and "$". Other examples of unusual characters from the data set include Greek letters in the middle of words, or the German character "ß" (probably a misread capital B).

OCR technology has, similarly to other technologies that rely on machine learning, seen vast improvements in recent years. The open source OCR library Tesseract for example, has consistently been improving since it was developed over 30 years ago. Their latest stable release came out in 2021, and it can be expected that this is much better than in 2015 when the NBDigital corpus was last updated.[7]

The Norwegian Language Bank graciously offered to do a re-OCR on the books that we had selected for our poetry data set. The first batch we received consisted of 193 books, where all the source scans were printed in Gothic font. See Figures 3.1[8] and 3.2[9] for two examples of what the source scans for this batch look like. Example 3.4 is the same excerpt of text, from the same book scan as example 3.3, but read with a newer OCR system.

[7]https://github.com/tesseract-ocr/tesseract
[8]https://www.nb.no/items/d799dfad1b1d3a36547061ae93857e5c?page=59
[9]https://www.nb.no/items/02f92e039e7bb56eb69ec83ebc72fccf?page=5

(3.4) Til
min Hustru.
W Ilk.er Errger og Glæder stemte
Min Harpe og gav den Klang,
Var det for dig, jeg digted
Og nynnede Sang efter Sang.

We observe that the quality of the text is much better, in this iteration most of the words are intelligible. Another improvement, which is crucial for the way we want to annotate rhyme, is that now the text is formatted as a stanza, with the rhyming words at the end of each line. This is also how it is formatted in the original book scan, which can be seen in figure 3.1. We can easily see that the 5th and 7th lines rhyme, with the line ending words being *Klang* (klɑŋ) 'chime' and *Sang* (sɑŋ) 'song'.

Even though we observe clear improvements in 3.4, there is still some noise, such as "W" and "Ilk.er". There are also examples where the new version is still mostly gibberish and impossible to comprehend, like example 3.5.

(3.5) »da faar du i Herrens vJoined j
jo Bingwr Mevand »gaj u9
»Da faar jeg 1 ivrneng Tenpel
jo Brug for mit Vievand!
Det er for godt for en Boler,
om end Jan er ndemand
Det er In * en voler
saa svarede Bispen varm;
»det sker ej, medens jeg ejer 5
en b i JoinedJoinedJoined

Figure 3.2 is the original page scan for this bit of text, and we can see that the pages are so thin that the letters from the other side of the page are visible. One might believe that this is the reason that the quality of figure 3.5 is so much worse than figure 3.4. What is interesting is that this text was actually much better in the original OCR, see example 3.6.

(3.6) "da faar du i Herrens Tempel jo Brug for dit Vievand."
"Da faar jeg i Herrens Tempel jo Brug for mit Vievand! Det er for godt for en Voler, om end han er Kongemand!
Det er for godt for en Voler, saa svarede Vispen varm; "det ster ej, medens jeg ejer en Livsgnist i denne Barm!

Though the formatting of the text is a bit worse with regards to line breaks, the text looks more or less perfect in example 3.6. There are no unexpected numbers or asterisks like in example 3.5. Notice the "Joined" on the first and last lines of example 3.5. This is some noise we believe is created by the OCR system, as it appears in several of the books from the first batch. These types of noise in the source data set call for data cleaning before annotation.

3.4 Norwegian? data

Another problem that became apparent during the compilation of our corpus was that the language used in the first batch was hardly Norwegian, but Danish (by today's standards, that is). In order to understand why that is, some insight into Norwegian history is needed, which is provided below.

3.4.1 A short overview of the recent history of the Norwegian language

In the period 1380–1814 (called *dansketiden*, 'the Danish time') Norway was in union with Denmark, or in the latter part (1537-1814) completely under Danish rule (Njåstad, Opsahl, and Scott, 2022; Weidling and Njåstad, 2022). During this period, the written Norwegian language was replaced by Danish. Almost all printed books were in Danish, and no books were printed in Norway until 1643, 200 years after Johannes Gutenberg invented the printing press (Venås, Gundersen, and Nordbø, 2022; Lehmann-Haupt, 2020).

Though the written language was Danish, it did not effect the spoken language for most people, as most people lived in sparsely populated areas where they used their different Norwegian dialects. In the cities, the vocabulary and pronunciation were somewhat influenced by Danish, especially among the higher classes (Venås and Nordbø, 2022).

Beginning with the rise of the national romanticism and *Det nasjonale gjennombrudd* 'the national breakthrough' in the 1840s, several changes were made to the written language. Asbjørnsen & Moe collected folktales from around the country and published them in a language that was close to how people actually spoke (Venås and Nordbø, 2022).

Henrik Wergeland and Knud Knudsen made efforts to "norwegify"[10] the written language, which was called *riksmål*. The first official distinction between Norwegian riksmål and Danish was made when the Ministry of Church and Education sent out a circular in 1862 that changed the spelling of several words. In 1866 *Retskrivningsregler til Skolebrug* 'Spelling rules for use in schools' was published by Jonatan Aars, and this and several later versions were used as the language norm for riksmål until 1907, when the first official spelling reform came out (Julien, 2021).

In contrast to the approach of norwegifying the already used written language (which was essentially Danish), Ivar Aasen wanted to replace it with a language that was purely Norwegian. In 1853 he presented *landsmål*, which was a written language based mainly on the Norwegian dialects that were closest to *gammelnorsk* 'old Norwegian', the language used in Norway from the beginning of the viking age to *dansketiden* (Nordbø and Magerøy, 2019; Bull, 2022).

In 1885 the Norwegian parliament decided that landsmål should be equal to riksmål. Each municipality could decide which language to use in schools, and in 1907 mandatory exams in both languages were introduced to Norwegian high schools (Venås and Nordbø, 2022).

In 1929, landsmål and riksmål were renamed to *nynorsk* 'new Norwegian' and *bokmål* 'book language'. The two written standards have undergone several reforms in the past century[11], reflecting the language policy of the time. During the 1900s, Norwegian language politics were influenced by the vision of *samnorsk* 'joint Norwegian': the hypothetical language that was the result of bokmål and nynorsk melting into one written language. In the language reform of 1938,

[10]*norvagisere:* https://snl.no/norvagisering
[11]Complete list here: https://snl.no/rettskrivingsreform

	Original OCR	Re-OCR 1	Re-OCR 2
Median year of publication	1890	1875	1921
Mean year of publication	1886	1875	1925
Min year of publication	1790	1814	1902
Max year of publication	2005	1905	1981
Total number of books	414	193	138

Table 3.2: Year of publication statistics for the different OCR-batches

several alternative forms were introduced to both nynorsk and bokmål, in an effort to make the two languages closer. In 1939, it was decided that radical bokmål was to be used in school books in the Oslo (the Norwegian capital) school district. During the 1950s this received a lot of backlash, and parents would correct their children's school books, as they thought the language used was vulgar. The samnorsk language policy was later dropped, and officially removed from Norwegian law in 2002 (Gundersen, 2022; Lars S Vikør, 2022).

It is important to note that both bokmål and nynorsk are written standards for the same language, and that there is no spoken standard for Norwegian. There are many dialects, and these two written forms do not capture all the variation in spoken Norwegian. Bokmål is most similar to *Østlandsk*, the dialects spoken in Eastern Norway, in and around the capital. Nynorsk is most similar to some dialects spoken in Western Norway (Lars S. Vikør, Jahr, and Berg-Nordlie, 2022; Venås and Skjekkeland, 2022). In Norwegian schools today, 87.3% use bokmål as their primary language, and 11.6% use nynorsk (Stolpe Foss, 2022).

3.4.2 Language and publication year distribution in the data set

Now, if we examine the year of publication for all the books we have extracted, the reason that the language is Danish is obvious. The median year of publication for the first batch of re-OCR is 1875, and all books are published between 1814 and 1905 (see table 3.2). This is well within the time when Norwegian riksmål was basically Danish.

Since the original data set contains books published up until 2005, we compiled a list of all the books that were published in 1900 or later, and asked specifically for a re-OCR of these, the intuition being that the language in those books were closer to contemporary Norwegian. Our list consisted of 165 books, or 148 if we subtract the ones we had already received re-OCR of. Of these 148, we received re-OCR for 138. Though these books are newer, the median year of publication is still 1921, meaning that most books are at least 100 years old.

Another interesting thing to note is the language distribution of the data set. Conveniently, the language codes mentioned in sections 3.2 and 3.2.1 also encode nynorsk and bokmål. As can be seen in table 3.3, there are a lot fewer books written in nynorsk than bokmål, with 7% of the data set being nynorsk compared to bokmål making up 92% of the data set. The four books with the

	Full data set	Pre 1900	Post 1900
Norwegian bokmål (nob)	380	242	138
Norwegian nynorsk (nno)	30	3	27
Norwegian (nor)	4	4	0
Total	414	249	165

Table 3.3: Norwegian language distribution in the data set

	Full re-OCR	Pre-1900	Post-1900
Norwegian bokmål (nob)	300	172	128
Norwegian nynorsk (nno)	20	3	27
Norwegian (nor)	1	1	0
Median year of publication	1897	1871	1920
Mean year of publication	1896	1872	1922
Min year of publication	1814	1814	1900
Max year of publication	1981	1899	1981
Total number of books	331	176	155

Table 3.4: Language and publication year distribution for the received re-OCR

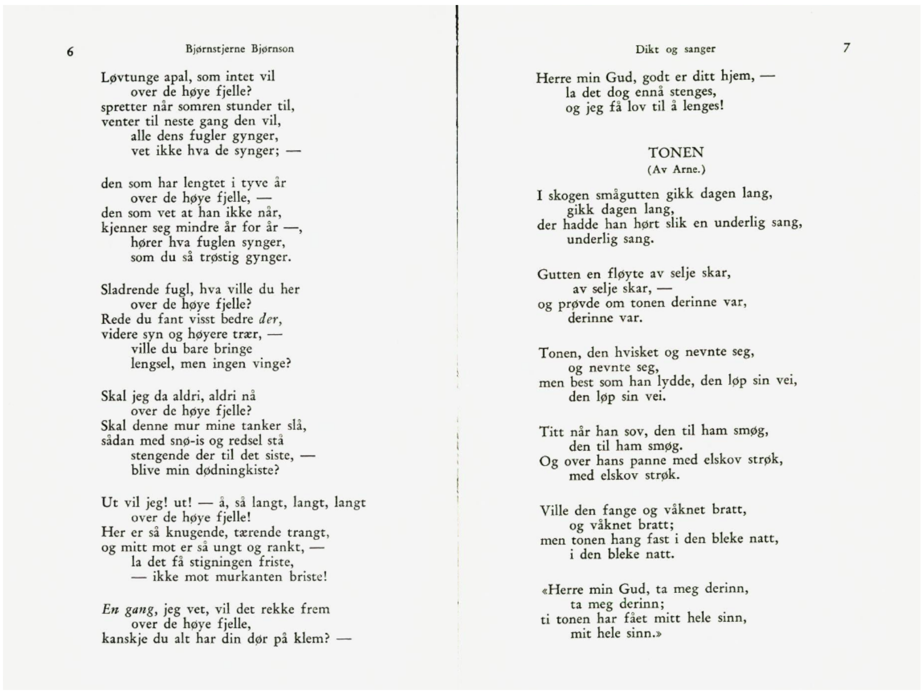
language code "nor" seem to be in bokmål.

If we divide the books into those published before 1900 and after 1900, we see that the proportion of nynorsk is different. Books in nynorsk make up 16% of the books published after 1900. The earliest nynorsk book in the data set is from 1877, which makes sense as the language was "invented" just a couple of decades earlier.

3.5 A public domain Norwegian poetry collection

In total, 331 books were re-OCRed for our corpus collection. We divided these into two groups, one consisting of the books published before 1900 and one of the books published in or after 1900. In table 3.4 the publication year and language statistics for the full data set and both groups are collected. The pre-1900 group consists of 176 books, and the post-1900 set 155. For the rhyme annotation, we will only go forward with the post-1900 group, as we are more interested in texts that are similar to the Norwegian that is written today. But, since these texts probably are of better quality than the source corpus, which we do not know when will be updated, we publish this whole data set so that anyone may use them. The data set can be found here[12].

[12]https://github.com/titaenstad/public_domain_poetry



6 Bjørnstjerne Bjørnson

Løvtunge apal, som intet vil
over de høye fjelle?
spretter når somren stunder til,
venter til neste gang den vil,
alle dens fugler gynger,
vet ikke hva de synger; —

den som har lengtet i tyve år
over de høye fjelle, —
den som vet at han ikke når,
kjenner seg mindre år for år —,
hører hva fuglen synger,
som du så trøstig gynger.

Sladrende fugl, hva ville du her
over de høye fjelle?
Rede du fant visst bedre *der*,
videre syn og høyere trær, —
ville du bare bringe
lengsel, men ingen vinge?

Skal jeg da aldri, aldri nå
over de høye fjelle?
Skal denne mur mine tanker slå,
sådan med snø-is og redsel stå
stengende der til det siste, —
blive min dødningkiste?

Ut vil jeg! ut! — å, så langt, langt, langt
over de høye fjelle!
Her er så knugende, tærende trangt,
og mitt mot er så ungt og rankt, —
la det få stigningen friste,
— ikke mot murkanten briste!

En gang, jeg vet, vil det rekke frem
over de høye fjelle,
kanskje du alt har din dør på klem? —

Dikt og sanger 7

Herre min Gud, godt er ditt hjem, —
la det dog ennå stenges,
og jeg få lov til å lenges!

TONEN
(Av Arne.)

I skogen smågutten gikk dagen lang,
gikk dagen lang,
der hadde han hørt slik en underlig sang,
underlig sang.

Gutten en fløyte av selje skar,
av selje skar, —
og prøvde om tonen derinne var,
derinne var.

Tonen, den hvisket og nevnte seg,
og nevnte seg,
men best som han lydde, den løp sin vei,
den løp sin vei.

Titt når han sov, den til ham smøg,
den til ham smøg.
Og over hans panne med elskov strøk,
med elskov strøk.

Ville den fange og våknet bratt,
og våknet bratt;
men tonen hang fast i den bleke natt,
i den bleke natt.

«Herre min Gud, ta meg derinn,
ta meg derinn;
ti tonen har fået mitt hele sinn,
mit hele sinn.»

Figure 3.3: Example of a book scan from the source data set

3.6 Pre-processing the data for annotation

Before we could go ahead and annotate the rhyme scheme of the poetry, some manual pre-processing was necessary. First of all, some of the books were not even poetry books, but books about poets. This is because we selected books based on a simple regular expression, and the Norwegian word for poem, *dikt* 'poem' is a substring of the word poet, *dikter* 'poet'. Of the books that did contain poetry, some of the poetry did not rhyme, but rather followed a more free structure. Thus, not all the books extracted could be used for rhyme annotation.

3.6.1 Removing noise

As mentioned previously, as the source files were built from automatically read book scans, they contain various amount of noise depending on the layout of the book pages. For example, most books have numbered pages. This means that page numbers periodically appear in the middle of the text throughout the source files.

If the book page layout also contains a header and/or a footer, this creates additional noise. See the book scan in figure 3.3. The page number and the author name are in the right page header, and the book title and page number are in the left header. Because the poems run over several pages, this text would appear in the middle of the poems in the text files created from the book scans. Removing this noise, in addition to the noise of OCR-misreadings, are parts of the necessary pre-processing for annotation. We did this manually.

3.6.2 Splitting books into poems and poems into stanzas

Another necessary pre-processing task during the creation of our corpus was to split the books into poems, and poems into stanzas. The word stanza here is used liberally, the way we split the books may not always be in line with what the author of the poetry intended. During annotation, there are only so many lines of pairwise rhymes the annotator can keep track of, so an effort was made to keep the stanzas short (but still contain lines that rhyme, of course).

This pre-processing step was done by manually going through each book text file, and adding beginning and end markers on either end of each poem, making sure that the stanzas and number of line breaks looked alright along the way. Some cleanup of OCR-misreadings was performed as well. After each book was processed, it was programmatically split up into poems based on the start and end markers, and each poem was written to file. In this step we also separate bokmål and nynorsk poetry. In the cases where a poem in nynorsk appeared in a book that was marked as bokmål or vice versa, this was also marked and sorted out.

3.6.3 "Norwegifying" the language

Some poems still seemed more Danish than Norwegian (recall that the median publication year of the source poetry is 1920). We want the annotated data set to be as similar as possible to contemporary Norwegian, and therefore we include this last pre-processing step: norwegifying the poetry.[13]

We use the intuition of a native speaker of Norwegian to identify the words that are no longer deemed Norwegian, and replace it with the modern Norwegian equivalent. Then, using the simple search-and-replace function that is available in modern editors, we replace these words in all files in the data set. Some examples are *efter* → *etter*, *mand* → *mann*, *mellem* → *mellom* and *bliver* → *blir*.

One problem with this approach is that it will inevitably disturb the rhyme scheme. One example from the data set is the following stanza:

(3.7) Nu må deres bundne sans
løses helt, så de herefter
kjenne kan mitt rikes krefter,
og kan se dets stråleglans.»

Line 2 and 3 of this stanza rhymes, with the line-ending words being *herefter* (hæreftər) 'henceforth' and *krefter* (kreftər) 'forces, powers'. However, the modern bokmål spelling of the former word is *heretter* (hæretːər) 'henceforth'. This means that modernizing this stanza removes a rhyme. In this case, our assessment is that it is okay that we lose the rhyme, because *herefter* is definitely Danish and not Norwegian.

[13] Because the poetry is no longer protected by copyright laws, this is not problematic legally, but we acknowledge that this step might be a little bit controversial. After this step, these texts are no longer the exact same poems that the authors wrote.

It is important to note that this step does not mean that we replaced every word in the texts that were not strictly bokmål. We have replaced most words that are distinctly Danish, but tried not to alter the poetry too much, especially line-ending words that may change the rhyme scheme.

Poetry is an art, and a genre of text that does not need to follow strict spelling and grammar, unlike for example news articles or scientific papers. A demonstration of this can be seen in the stanza in example 3.8. The words "*ælva*", "*tåli*", "*når'n*", "*dråpan*" and "*me*" are not accepted in bokmål, the "correct" forms would be *elva* 'the river', *tålelig* 'bearable', *når man* 'when one', *dråpene* 'the drops' and *med* 'with'.

(3.8) Ombord i en husbåt i ælva
kan livet bli tåli bra,
når'n bare har dråpan på flaska
å smøre me nå og da.

This spelling is of course an artistic choice, but also works as a guidance for how to pronounce the words when reading this stanza aloud[14], as they are closer to spoken language.

Though this step is described as a pre-processing step (and ideally should be), we continued to discover words that needed to replaced during annotation, and thus it was done simultaneously.

3.7 Annotation

In order to annotate the rhyme scheme of the poems, we created an annotation script that can be found on github[15]. The main annotation effort was performed by the author, however, to ensure high quality annotations we had two annotators doubly annotate a subset of the data, and measured the inter-annotator agreement using Cohen's kappa.

To help the annotators create consistent annotations, annotation guidelines were created, which the annotators were required to read through before starting the annotation process. The guidelines further serve to describe the corpus annotations in more detail and will be distributed along with the corpus. We present the annotation guidelines below.

3.7.1 Annotation guidelines

For each stanza, the annotator should provide a rhyme scheme code; a string of characters of the same length as the number of lines in the stanza. The characters in the rhyme scheme code should correspond to the rhyme scheme of the stanza. If the first and second line of the stanza rhyme, the first and second letter of the rhyme scheme code should be the same letter.

For the stanza

[14] As some say, poetry should be heard and not read https://www.theguardian.com/books/booksblog/2006/oct/31/thespokenwordwhypoetrysho

[15] https://github.com/titaenstad/norwegian_rhyme_scheme_corpus/tree/main/annotation_tool

(3.9) Den natten bygget vi et drømmeslott
med hvite saler, blanke marmorflater
det skulle ligge høyt og sees godt
i sol og luft med åpne, lyse gater

A correct rhyme scheme code is `ABAB`. Examples of incorrect rhyme scheme code are `ABBA`, `AAAA` or `AABBA`.

Acceptable characters for rhyme scheme code

You may use any of the characters `QWERYUOPASDFGHJKLZXCVBM` to annotate the rhyme scheme of a stanza.

Special characters

There are three special characters that are used to annotate lines that for some reason cannot be annotated for rhyme scheme normally.

T - Title

Used when:

- the line is the title of the poem.

If the data is formatted correctly, this should usually be on the first line of the first stanza of the poem. Not all poems contain a title.
For the stanza in example 3.10 a correct rhyme scheme code is `TABAB`.

(3.10) RIDDERSLAG
Vi har en verden å vinne
og ikke en time å miste.
Så skulle vi famle i blinde
og spikre oss selv en kiste?

I - Info

Used when:

- the line is not a part of the poem itself, but contains information about the poem.

For the stanza in example 3.11 correct rhyme scheme code is `TIABAB`.

(3.11) ENGELSKE SOCIALISTER
Etter generalstreiken.
Nå vask deres hender! Ta på dere jakken,
Mac Donald og Thomas og Henderson!
Nå ligger han rolig med kniven i nakken,
den tosk! La ham bare bli liggende sånn.

N - Noise

Used when:

- the line is not in Norwegian or most words are noisy/unintelligeble
- the line-ending-word contains noise/is not a proper word

For the stanza in example 3.12 a correct rhyme scheme code is `NNABBCCB`, because lines 1 and 2 contain too much noise for us to want to include it in our dataset, even if the line-ending-words are free from noise.

(3.12) fo ^{6}) pndehe s ro ^{7}ft fjerner,
$_{88}$ er mantdøde fiksidders rov,
og trærne ser man ei for bare skog,
man knuser nøttene med skall og kjerner.
En patriot forteller at han verner
om landets kraft når bare han er grov,
og vraker åndens nektar med en lov
av tankefinkel fra de tomme hjerner.

The definition of rhyme

Sometimes there are stanzas containing word pairs that do not fully rhyme, but almost. Because the data is fairly old, you might observe examples where the words would rhyme if you replace the t-sound with the d-sound or other ways to speak that are similar to old riksmål. Try to annotate the rhyme as accurately as possible given how the stanza is written.

For the stanza in example 3.13 a correct rhyme scheme code is `ABCB`.

(3.13) Verden, hvor er du full av gåter,
kverner som hav i klippering,
maler din must til lette fråder,
knuser din kraft til ingenting!

It might be tempting to annotate "gåter" (gɔtəɾ) and "fråder" (fɾɔdəɾ) as rhyming, as they are indeed similar (and this was probably what the author meant as well, when this was written almost 100 years ago). But, because we want the data to reflect modern (Østlandsk) Norwegian, please do not annotate word pairs as rhyming if they do not rhyme the way you would pronounce it. We do not want to *danify* our language.

For the stanza in example 3.14 a correct rhyme scheme code is `AABCCB`.

(3.14) Her ser du i blest bak det vimrende løv
i sortmalte linjer: Guds fred med ditt støv!
hvor sorgen sitt minnekvad satte.
Om enkelt og kluntet, mitt hjerte hvor rikt,
når livet fortolker de skjønneste dikt
som hjertene bare kan fatte

Though "rikt" (ɾiːkt) and "dikt" (dikt) do not rhyme 100%, with the "i" being long in "rikt" and short in "dikt", the words are similar enough that this should be accepted as rhyme. As a rule of thumb, using long/short vowels is a lesser rhyme crime than completely different consonant/vowel sounds.

Disagreeing rhyme schemes	Kappa on stanza level	Kappa on line level
6/100	0.926	0.959

Table 3.5: Cohen's kappa between the annotators of the subset

3.7.2 Inter-annotator agreement

In order to assess whether it is possible to annotate rhyme consistently and in an objective manner, we have doubly annotated a subset of the data. Because the whole data set was already annotated by the main annotator, we were able to select the subset based on rhyme pattern frequencies. We chose 100 stanzas from the ten most frequent rhyme patterns in the data set, weighted so that the number of stanzas chosen is proportional to the frequency of the rhyme scheme.

We measure the agreement of the two annotators with Cohen's kappa. Cohen's kappa is used for categorical annotation, when each item is annotated with **one** label from a set of mutually exclusive labels. The formula is used for calculating agreement between two annotators, and it takes the possibility of agreement by chance into consideration. It is given as:

$$K = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{3.15}$$

where Pr(a) is the observed agreement, and Pr(e) is chance agreement. The kappa ranges between -1 and 1, where the higher the number the more agreement, and a score of 0 or lower means no agreement. (McHugh, 2012).

We measure agreement on two different levels. One is on stanza level: we let each stanza be an item, and the rhyme scheme code the label. This means that the rhyme scheme code for a stanza must be an exact match for the annotators to agree. The other is at the line level, where each line is an item, and the rhyme scheme letter for that line is the label. Then the observed agreement for annotations `AABB` and `AABC` will be 3/4, whereas `AABB` and `ABCD` will be 1/4. With the former way of measuring both would be 0.

In table 3.5 the results of the double annotation of the subset are presented. Out of the 100 stanzas, there were 6 stanzas where the two annotators' annotations were different. Of these, 3 were examples where the stanza contains an almost-rhyme, and 3 where examples where the rhyme scheme was wrongly annotated by the main annotator. One such example is 3.16 below. The rhyme scheme in this stanza is obviously ABBA, but it was wrongfully annotated with ABAB by the main annotator.

(3.16) I den falt gnisten av min moders lyse
og varme sjel; og ekteskapets lykke
ble ei alene deres alders-smykke;
men den vil lenge etter døden lyse.

Still, the Cohens' Kappa score is very high for both stanza level and line level agreement. This shows that the annotated data set in general should hold a relatively high quality, and be mostly correctly annotated according to the guidelines. For future work, doubly annotating more of the data set to further ensure correct rhyme schemes would be a good idea.

3.8 Challenges

Producing the data set was very time consuming. In particular, the pre-processing step of splitting books into poems and stanzas took a lot of time, because we also removed noise during this step. The pre-processing was actually more time consuming than the annotation itself. Ideally we would have been able to go through all the books, but we did not. We started with the bokmål books, and therefore the whole produced data set is bokmål only, though it would be very interesting to annotate a nynorsk data set in the future. We view the current annotation effort as a proof-of-concept. Since the annotation guidelines and source data have been made available, the annotation effort can easily be extended in the future.

3.8.1 Choosing a dialect

Annotating phonetic information from text, when there is no spoken standard in Norwegian is a big challenge, and there is arguably no objective truth. A word pair that rhymes in one dialect may not rhyme in another. Unless the poetry is obviously written in a specific dialect, such as example 3.8, for the texts marked with bokmål we assume the Østnorsk dialect that is spoken in and around the capital. We choose this because it is the dialect that is most similar to the written bokmål, and the dialect that the annotators speak. More than half of the population lives in Østlandet, and 40% live in the greater Oslo area (Thorsnæs, 2021). Thus the chosen dialect should be fairly representative of spoken Norwegian.

3.8.2 Challenges during annotation

There were two specific cases that proved to be challenging in the annotation process. One is the case when the stanza consists of more that 6 lines. It took more than double the time to annotate a long stanza, because you need to keep track of all the previous lines.

The other case is when there is an *almost-rhyme*. One would think it would be pretty straightforward whether or not two words rhyme, but in reality these edge-cases could sometimes be very challenging to assess. On the one hand, we want to balance faithfulness to the author's intention with the purpose of the corpus creation, which was to provide training data for rhyme modelling. On the other hand, we do not want our data set to contain incorrect rhymes.

3.9 Presenting NoRSC: Norwegian Rhyme Scheme Corpus

The final data set can be found here[16]. In total 11 books were annotated. This resulted in 508 poems, 5158 stanzas and 26198 lines of data (see table 3.6).

[16]https://github.com/titaenstad/norwegian_rhyme_scheme_corpus

# of books	11
# of poems	508
# of stanzas	5158
# of lines	26198
# of lines that potentially rhyme	25749
# of unique line ending words	6990

Table 3.6: Data quantity in the annotated data set

Rhyme scheme	ABAB	ABCB	AABB	AABCCB	ABBA
Count | % of total	1142 | 22%	749 | 15%	611 | 12%	360 | 7%	181 | 3,5%
Rhyme scheme	AABBCC	AAA	ABAAB	AABCBC	ABABCC
Count | % of total	162 | 3%	152 | 2.9%	73 | 1.4%	70 | 1.4%	58 | 1.1%

Table 3.7: Frequency of the top 10 most frequent rhyme schemes

3.9.1 Rhyme scheme statistics

We calculated rhyme scheme frequencies for the data set excluding the info, noise and title annotations, as lines annotated with these are not part of the rhyme. If a stanza of length 5 is annotated `TABAB`, then we count it as `ABAB` and so on. There are 449 lines in these categories, meaning that we have 25749 lines of potential rhymes.

The most frequent rhyme scheme in the data set is `ABAB` with 1142 stanzas, or >22% of the stanzas in the data set. After that comes `ABCB` with 749, or 15% of the stanzas (Table 3.7). There are 490 different rhyme schemes, but 303 of them only appear once, and the "long tail" distribution, typical of linguistic data, is apparent (see Figure 3.4).

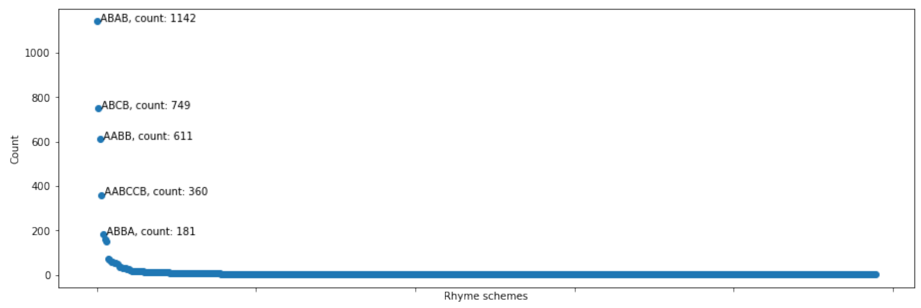


Figure 3.4: Rhyme scheme distribution in the annotated data set

3.10 NoRSC v1.1

As discussed in section 3.7.2, some of the stanzas were annotated with the wrong rhyme scheme. Going through all 5000 stanzas and checking the annotations is not a viable option. In chapter 4 we extract unique rhyme pairs from the annotated data set. From the "bad" rhyme pairs we were able to efficiently find stanzas that have wrong annotations. 133 stanzas were re-annotated to improve the quality of the corpus. 89 stanzas received a new rhyme scheme code. Both the original corpus discussed in this chapter, and the new and improved version can be found on the corpus github[17].

[17] https://github.com/titaenstad/norwegian_rhyme_scheme_corpus

Chapter 4

Rhyme pair collection

In this chapter, we extract both positive and negative rhyme pairs from the NoRSC data set, in order to use them for rhyme modelling in the next chapter. We also discover a graph based method to harvest even more rhyme pairs from the same number of stanzas. To get even more data, we scrape rhyme buckets from Wiktionary, from which we are able to construct even more positive and negative rhyme pairs.

As explained in chapter 2, a rhyme pair is a pair of words or words sequences that rhyme. Similarly, a negative rhyme pair is a pair of words or word sequences that do not rhyme. Rhyme pairs are the basis for rhyme modelling in Lau et al. (2018), Haider and Kuhn (2018) and Minogue (2021), as discussed in chapter 2. We also extract rhyme buckets, sets of words where all words rhyme.

4.1 Extracting rhyme pairs from NoRSC

We extract rhyme pairs from our rhyme annotated data set by combining every pair of line ending words annotated with the same letter in a stanza. For instance, from the rhyme scheme code `ABAB` we get two rhyme pairs. From the rhyme scheme code `AAA` we get three rhyme pairs. But how do we find out how many rhyme pairs we get from an arbitrary rhyme scheme? It is quite simple if we apply some simple graph theory.

We can view a stanza as a graph: We let each line in the stanza represent a vertex. The rhyme relation between two lines of poetry is represented as an edge between the two corresponding vertices. Rhyme is a symmetrical relation, A rhymes with B means that B rhymes with A. Therefore the graph is undirected.

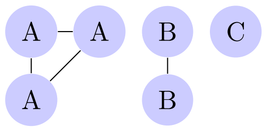


Figure 4.1: Graph illustration of the rhyme scheme `ABABAC`

This graph can be constructed using only the rhyme scheme code: Let each symbol in the rhyme scheme code be a vertex. Place an edge between every pair of vertices with the same symbol. See Figure 4.1 for an example of the *rhyme scheme graph* created from the rhyme scheme code `ABABAC`. There are 6 vertices, one for each letter in the rhyme scheme code. There are four edges, one for each rhyme pair that can be constructed from the stanza with the annotation.

Now, in order to find the number of rhyme pairs for any rhyme scheme, we simply need to count the number of edges in the graph constructed from the rhyme scheme. Making the graph and counting the edges manually is tedious and unnecessary, as this can easily be calculated.

We first observe that the rhyme scheme graphs have the following properties:

1. The graph is separated into *connected components*, the connected subgraphs of the larger graph. There is one component for each unique rhyme scheme symbol.

2. Each component contains as many vertices as occurrences of the rhyme scheme symbol it represents.

3. Each component is a *complete graph*; a graph where each vertex has an edge to every other vertex

Knowing these three facts, we can calculate the number of edges in any rhyme scheme graph, thereby also calculating the number of rhyme pairs. The number of edges in the graph would be the same as the sum of the number of edges in each component. Each component is as we know a complete graph. To calculate the number of edges in a complete graph, we use the following equation:

$$E = \frac{V(V-1)}{2} \tag{4.1}$$

where E is the number of edges, and V is the number of vertices. $V(V-1)$ because every vertex has an edge to every other vertex in the graph. The division by 2 is because the graph is undirected; the edge (A,B) is the same edge as (B,A).

Now we have all the pieces needed to create algorithm for counting the number of rhyme pairs for a given rhyme scheme. For each unique rhyme scheme symbol s, there is a complete graph with the same number of vertices as occurrences of s. Summing up the number of edges for each of these complete graphs, we get the total number of rhyme pairs for the given rhyme scheme. See listing 1 for an implementation of this.

Using the code in listing 1 on our entire annotated data set, we get a total of 12 627 rhyme pairs. We have no guarantee that these are unique, though. As seen in table 3.6 in chapter 3, the number of unique line ending words in the data set is just 27% of the number of potentially rhyming lines. Extracting all rhyme pairs and removing duplicates, we get 7238 unique rhyme pairs, 57% of the total number of rhyme pairs.

We also extract negative rhyme pairs. We want to be careful with almost-rhymes, as we fear that these might confuse an eventual rhyme model. Because there is a high possibility of almost-rhymes for the stanzas with rhyme codes such as `ABCB`, we only use stanzas where the number of symbols in the rhyme scheme code was equal for all symbols. This was true for 3006 stanzas. For all

```python
from collections import Counter

def get_number_of_edges(vertices: int) -> int:
    return (vertices*(vertices-1))//2

def number_of_rhyme_pairs(scheme: string) -> int:
    c = Counter(scheme)
    return sum(get_number_of_edges(v) for v in c.values())
```

Listing 1: Python implementation of counting rhyme pairs for a given rhyme scheme

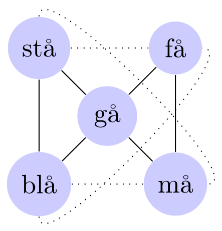


Figure 4.2: Graph illustration of “gå” as a “bridge” vertex.

these stanzas, we pair up all line-ending words with different annotations. This resulted in 22 447 unique negative rhyme pairs.

4.2 Graph based approach to collecting rhyme pairs across stanzas: dense rhyme pairs

It is possible to extract even more rhyme pairs from the annotated stanzas. We expect rhyme to be a transitive relation: if A rhymes with B and B rhymes with C, we expect A and C to rhyme too.

Let us say we have a stanza where “blå”, “gå” and “stå” appear as line-ending words. This gives us 3 rhyme pairs. From another stanza we have the rhyming words “gå”, “få” and “må”, also giving us 3 rhyme pairs. This is a total of 6 rhyme pairs for both stanzas. But because “gå” appear in both stanzas, we can expect that all five words rhyme.

Viewing the line-ending words as vertices, and the rhyme-relation as edges, it is clear that the “gå” vertex connects the two stanzas’ graphs. When a word vertex connects two rhyme groups like this, we call it a *bridge word* (see Figure 4.2). Now, instead of two graphs with 3 vertices, we have one graph with 5. In Figure 4.2 the continuous lines are the already existing rhyme pairs, and the dotted lines represent the added rhyme pairs we get from merging the two graphs. This is also confirmed by using equation 4.1, we see that one complete

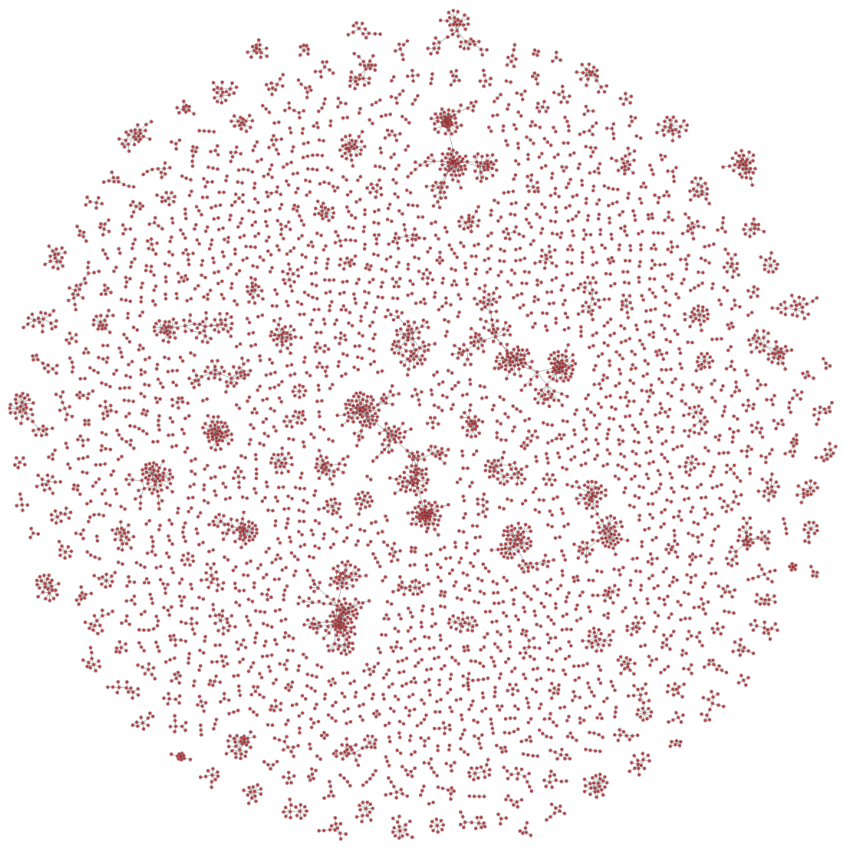


Figure 4.3: Graph representation of the rhyme pairs; Each vertex is a word, and each edge is a rhyme pair

graph of 5 nodes has 10 edges, 4 more than two complete graphs of 3 nodes. One important property of complete graphs, as seen in equation 4.1, is that the number of edges increase quadratically as the number of vertices increase.

In Figure 4.3 we see all 7238 rhyme pairs represented as edges in a graph. The vertices are the unique line-ending words, 6290 in total. This graph is separated into 1259 connected components. Here the vertices in every connected component amounts to a “bucket” of words that all rhyme with each other. If we combine every pair of words for each or these buckets, we get a lot more rhyme pairs. This is equivalent to adding edges such that every connected component becomes a complete graph. We call these rhyme pairs *dense*, because we obtain them by making a more dense graph out of the rhyme pair graph.

4.2.1 Connected components and consequences of wrong annotations

Of the 1259 connected components, 727 have only two vertices, meaning that they are simply isolated word pairs. These pairs are already complete graphs, and we can obviously not get any more rhyme pairs out of them. Of the 532 remaining components, the average number of vertices is 9. In comparison the average number of lines in a stanza is 5, which means that many rhymes were connected across stanzas. Note that there is no guarantee that all words that rhyme have been connected in the same component. This is completely dependent on there being *bridge words* that connect rhyme pairs across stanzas.

Upon inspecting the words in each bucket it becomes clear that not all of these can be used as they are. In section 3.7.2 we discover that 3 out of the 100 doubly annotated stanzas had received a wrong rhyme scheme annotation. For every wrong rhyme pair, two components are connected that should not be connected. If two line ending words that are frequently used are wrongfully annotated to be rhyming, this would connect two very large components that should not be merged. We can see in Figure 4.3 that the largest components seem to contain several smaller clusters that are connected by only a few edges.

When only sampling rhyme pairs directly from the stanzas, a few incorrect rhyme pairs will not impact the data set quality too much. But with dense rhyme pairs, that is not the case. The largest connected component produced with our rhyme pair data set contains 227 vertices, making up a bucket of words with at least 8 different phonetic endings (see Figure 4.4). A complete graph with 227 vertices has 25 651 edges. If we were to create all possible rhyme pairs from this single bucket, we would end up with 25 651 word pairs, but most of them would not actually rhyme.

In order to assess the quality of the derived graph, we manually inspect the resulting buckets. It seems that all buckets that contain less than 10 words have no errors. We look over all the larger buckets, and find that 27 buckets contain words that do not rhyme. 505 of the buckets (of size >2) are good to use as they are. These 505 "good buckets" have an average size of 6.6, and contain 3327 unique words, 52.9% of the total vocab (aka the number of vertices in the graph). We construct rhyme pairs by combining all possible pairs of two words within each bucket. This results in a total of 17 706 unique rhyme pairs. 1941 of these already exist in the original rhyme pair set. This means that the number of rhyme pairs was increased with 15 765 dense rhyme pairs.

4.2.2 Two approaches to repairing the bad components

We do not want to lose the words from the "bad" connected components, especially because all the 9 largest connected components in the rhyme pair graph are among these. We use two approaches to split these components into the separate appropriate clusters.

Clustering with HCS algorithm

One approach is to use a graph clustering algorithm to separate the connected components. We use the Highly Connected Subgraphs (HCS) algorithm from

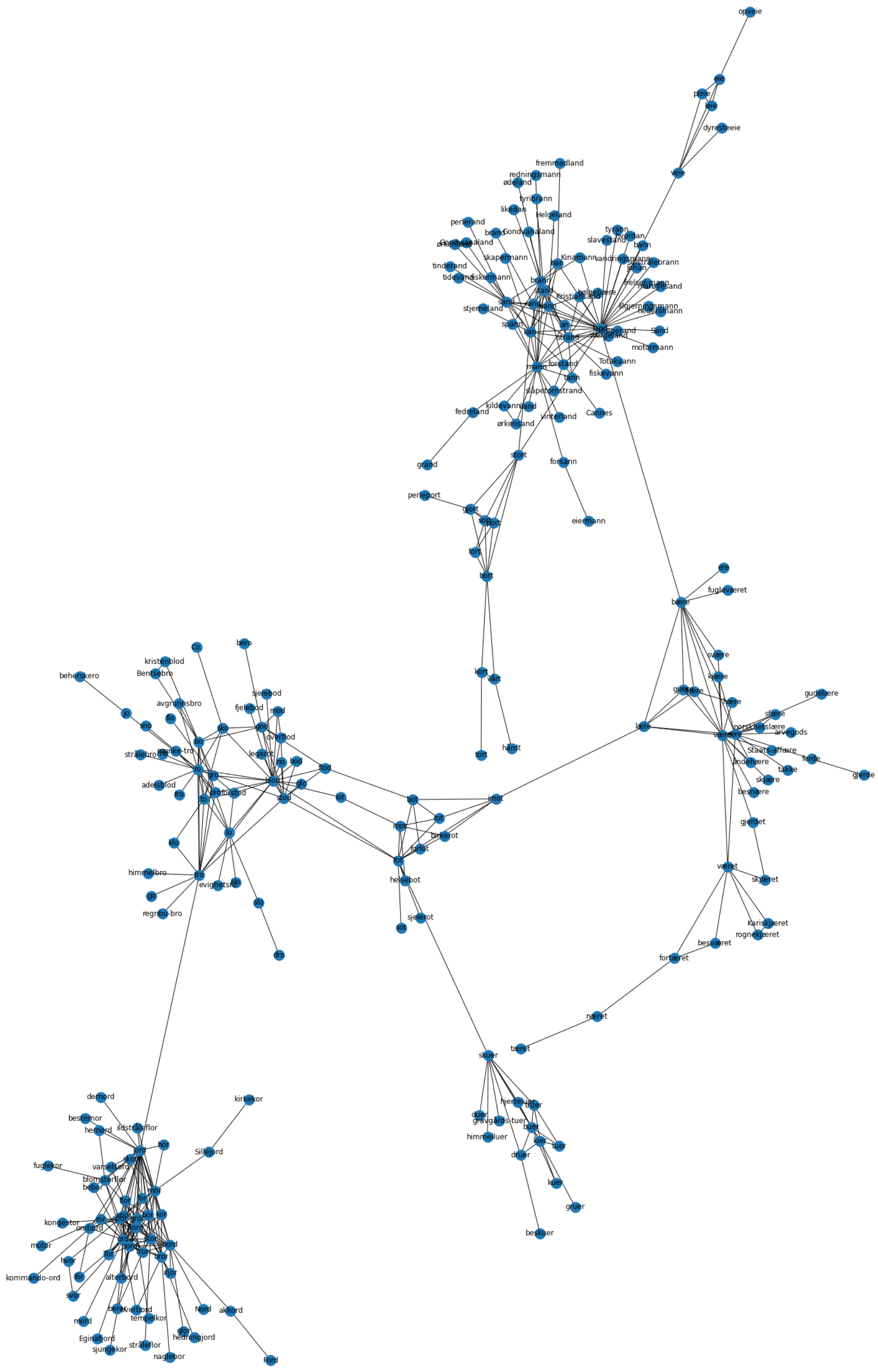


Figure 4.4: The largest connected components from the rhyme pair graph

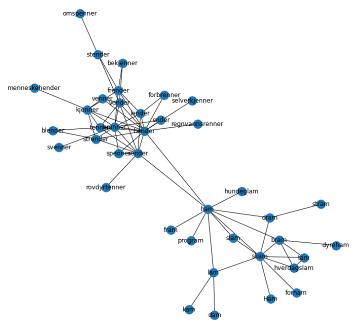


Figure 4.5: Connected component before clustering

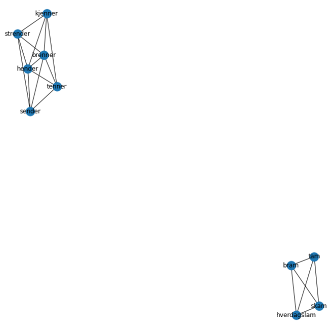


Figure 4.6: Connected component after clustering

Hartuv and Shamir (2000), implemented in python by Gert Sluiter[1]. The algorithm takes an undirected, connected graph as input, and outputs a graph that has been split into several highly connected graphs. The reasoning for using this algorithm is that we expect each component to have more edges between vertices of words that actually rhyme, and fewer edges between vertices of words that do not rhyme, and were added by accident.

In Figure 4.5, the 7th largest connected component is displayed. Figure 4.6 shows the result of running this component through the HCS algorithm. The graph has been separated into two components, with words ending in ɛnːər (top left) and ɑm (bottom right). 29 of the 39 vertices in the original graph have been separated into single-vertex components (not displayed in the figure).

The results of running each of the 27 “bad” components through the HCS algorithm is as follows: 6 components were not separated into several graphs. The other 22 were separated into 70 components of size >1, containing a total of 336 words. 1055 vertices were separated into single-vertex components, meaning that the corresponding 1055 words could not be used to create more word pairs. From the 70 word buckets that were extracted, the average number of words is 4.8. For each bucket, rhyme pairs were constructed from all possible word pair combinations. This resulted in 787 rhyme pairs, where 338 already existed in the original rhyme pair data set. This means that using the HSC algorithm, the number of rhyme pairs was increased by 449.

Manual clustering

The other approach is to manually go through each component, and remove the edges that should not be there. This is approach is motivated by the fact that the graph clustering algorithm left so many unconnected vertices, and that it seemed pretty easy to identify the problem edges manually. For example, in the 7th largest component (Figure 4.5), we removed the edges (“ham”, “lender”) and (“ham”, “sender”). This left two separated components of 17 and 22 vertices.

[1]https://github.com/53RT/Highly-Connected-Subgraphs-Clustering-HCS

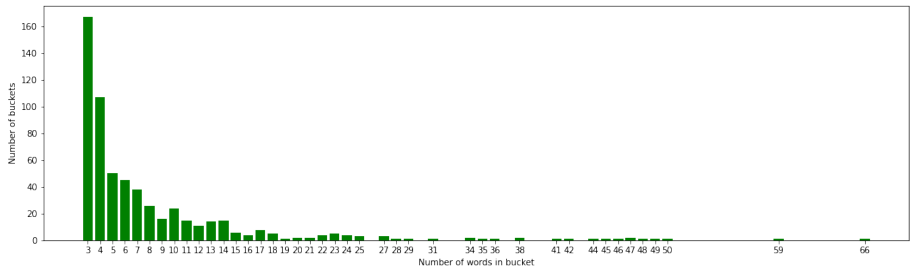


Figure 4.7: Words per bucket for the good buckets + manually separated rhyme buckets

In comparison, using the clustering algorithm we were left with two components of 6 and 4 vertices, respectively. The advantage with using the algorithm is of course that it can be done automatically, but as we have already spent time manually annotating the data set, we took the time to manually remove problematic edges from the 27 bad components as well.

From the manual clustering we discovered that it was not only mistakes during annotation that led to non-rhyming words being connected in the rhyme pair graph. There were several problematic edges that were not a result of annotation mistakes. *Heteronyms*, words that are spelled the same, but have different meaning and pronunciation, were the source of several problematic edges. One example is the letter sequence “sort”, that means ’black’ if pronounced suʈ, but ’kind, type’ if pronounced sɔʈ.

A related case is words with multiple possible pronunciations. The word *tid* ’time’ can be pronounced both tiː or tiːd, even by the same speaker.

The problem edges that resulted from wrong annotations can be separated into two cases. The first case is from obviously wrong rhyme scheme annotation: the words in the rhyme pair are not phonetically similar at all. The other is almost-rhymes, word pairs where the words *almost* rhymes, but not complete, and have wrongfully been annotated as rhyming.

In total 106 edges were removed, which separated the 27 graphs into 91 components. The average number of vertices per component is 16.4, and they contain 1492 words in total. 16 vertices separated into single-vertex components, and could thus not be used to create more rhyme pairs. All word combinations from all buckets resulted in 20 183 rhyme pairs. Of these, 1263 already existed in the original rhyme pair data set. The number of rhyme pairs was increased by 18 920.

In 11 out of 27 cases, the clusters created by the HCS algorithm agree with the manual clustering, with regards to the phonetic word endings in the different clusters and the number of clusters. Generally the HCS algorithm created fewer clusters than the manual clustering. The clusters also had fewer words, which in turn yields fewer rhyme pairs. Also here the fact that the number of edges is quadratic to the number of vertices in a complete graph shows, with the 449 new rhyme pairs from the HCS clustering versus the 18 920 new rhyme pairs from the manual clustering.

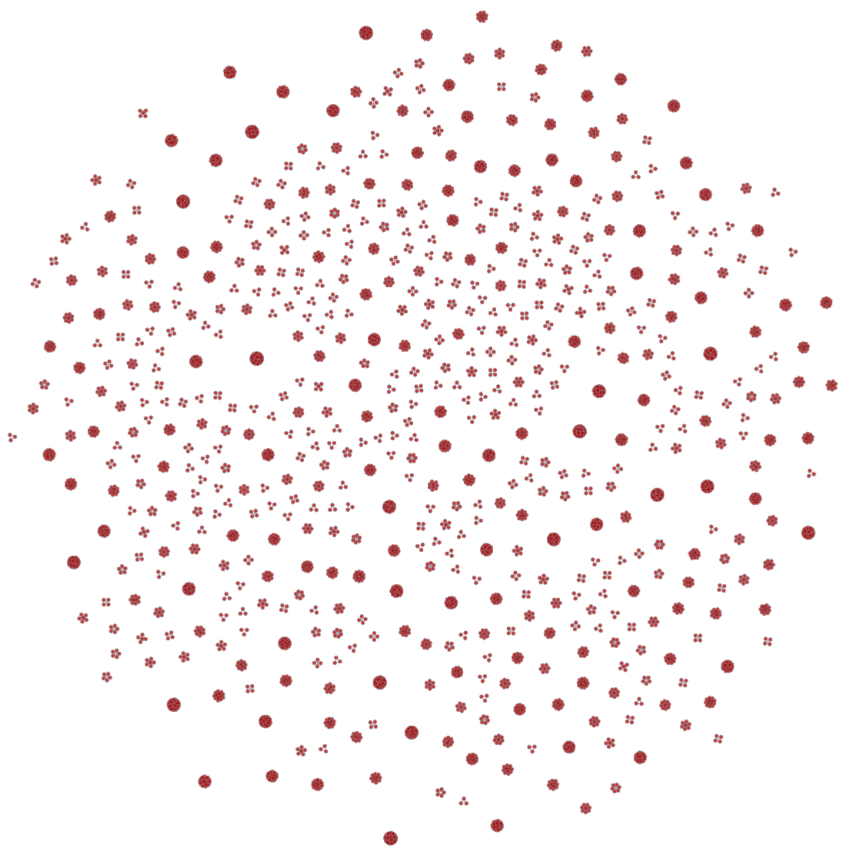


Figure 4.8: Graph representation of all rhyme pairs after manual clustering

Figure 4.7 shows the size distribution of the buckets produced from the manual clustering. We see the long tail distribution, the smallest buckets are most frequent. 166 buckets contain 3 words, and 106 buckets contain 4 words. There are only 1 or 2 buckets for each size >28.

See appendix A for descriptions and figures of the graph clustering of all 27 “bad” components. Figure 4.8 shows the graph representation of the dense rhyme pairs: these include the rhyme pairs from the good buckets plus the rhyme pairs from the manual clustering.

4.3 Extract rhyme pairs from wiktionary

We also extract rhyme pairs from wiktionary, where there is a page called “Category:Norwegian rhymes”[2], containing links to 33 pages listing words with the

[2]https://en.wiktionary.org/wiki/Category:Norwegian_rhymes

#	Selection	Pos. rhyme pairs	Neg. rhyme pairs	Unique words
1	Original pairs from rhyme scheme annotated stanzas	7238	22 447	6290
2	Pair components	727		1454
3	Good buckets + 2	18 416		4780
4	HCS + 3	19 203		5116
5	Manual clustering + 3	38 599		6272
6	Wiktionary	80 363	849 993	1384
7	Wiktionary and 5 merged	155 838		7541

Table 4.1: Number of rhyme pairs extracted from different selections and buckets

same phonetic ending. As these words are already sorted into buckets, we just need to combine them into pairs. There were in total 1502 unique words in these 33 buckets, with an average size of 45 words per bucket. By combining all possible word pairs within each bucket, we got a total of 80 363 rhyme pairs.

Using the wiktionary rhyme groups we could also create negative rhyme pairs. With the groups (aka connected components) we constructed from our own annotations, we had no guarantee that two groups of words did not rhyme. This is not the case with the wiktionary buckets. Because wiktionary differs between short and long vowels, and we want to avoid marking almost-rhymes as negative, we first merge 6 pairs of buckets before we begin. From the remaining 27 buckets, we create negative pairs by pairing each word from each bucket with every other word from every other bucket. This resulted in 877 429 negative rhyme pairs.

4.4 Merging Wiktionary and dense rhyme buckets

Of the 1384 unique words from the Wiktionary buckets, and 6290 words from the NoRSC rhyme pairs, 114 words exist in both. As we know, larger buckets give more rhyme pairs, and we therefore merge the buckets from wiktionary and NoRSC. We use the good buckets + manually clustered buckets, and simply iterate over the overlapping words and merge the buckets from both data sets along the way. This resulted in a total of 612 buckets, with an average size of 10 words per bucket. The median bucket size is 5, due to the large number of buckets with 3 and 4 words in the good buckets + manually clustered buckets (see Figure 4.7). The 10 largest buckets contain between 50 and 324 words. From the merged buckets 155 069 rhyme pairs were constructed. See Table 4.1 for an overview of the number of rhyme pairs and total vocabulary from the different selections and buckets.

Chapter 5

Rhyme modelling

In this chapter we explore two types of rhyme modelling: rhyme prediction and rhyme generation. Both can technically be called rhyme classification: the first task is a binary classification task to predict whether two character sequences rhyme, and the second is to predict which rhyme bucket (aka class) an input word belongs to. In the former task, we base our model on, and compare our results to prior work. The latter task is designed by us such that models trained for this task can be used in poetry generation.

5.1 Rhyme prediction

Inspired by the "Supervised Rhyme Detection with Siamese Recurrent Networks" paper (Haider and Kuhn, 2018), we explore how well a siamese recurrent network (SRN) trained on character vectors performs rhyme detection in Norwegian. We were not able to find the code for Haider and Kuhn's experiments, we did however find a blog post titled *Using Siamese neural networks to create a simple rhyme detection system*, demonstrating a very similar experiment (Minogue, 2021). Haider and Kuhn (2018) gets slightly better results with ac curacies of 0.96 (German) and 0.97 (English), compared to the 0.95 accuracy on English data from Minogue (2021). The main differences are that Haider and Kuhn (2018) use less data, but a more complex model architecture, and Minogue (2021) use a lot more data, with a simpler model architecture. The details of both experiments are described in section 2.5 in chapter 2.

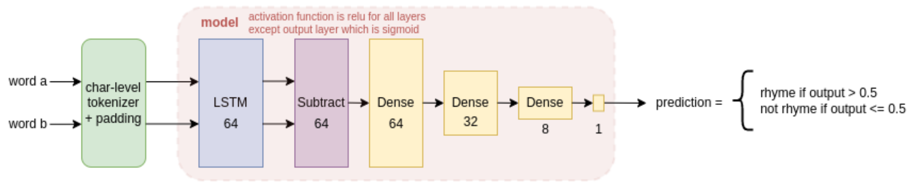

model
activation function is relu for all layers except output layer which is sigmoid
word a
word b
char-level tokenizer + padding
LSTM 64
Subtract 64
Dense 64
Dense 32
Dense 8
1
prediction =
rhyme if output > 0.5
not rhyme if output <= 0.5

Figure 5.1: Overview of the model architecture of the rhyme detection model

	Train set	Validation set	Test set	Total
unique word pairs	8685	1448	4343	14 476

Table 5.1: Data set split

```python
def last_letters(word: str) -> str:
    for i in range(len(word)-1, -1, -1):
        if is_vowel(word[i]):
            break
    return word[i:]

def rhymes(word_a: str, word_b: str) -> bool:
    return last_letters(word_a) == last_letters(word_b)
```

Listing 2: Python implementation of baseline rhyme detection

As the code for the experiments from Minogue (2021) is readily available[1], we use this for our rhyme detection model. See Figure 5.1 for an overview of the model architecture. The model takes in two words as input, and outputs a number between 0 and 1. If the output is more than 0.5, the prediction indicates that the words rhyme.

We start out with the original non-dense NoRSC rhyme pairs (selection 1 in Table 4.1). Following Minogue (2021), we employ a 1:1 ratio of positive to negative pairs, so we do the same. We use all 7238 positive pairs, and randomly sample the same number of pairs from the larger set of negative rhyme pairs. This makes up a total of 14 476 word pairs, or 1.4% the number of pairs as in Minogue (2021). Following Minogue (2021), the data is split into 60% training data, 10% validation data and 30% test data, keeping the ratio of positive to negative pairs in each set (see Table 5.1).

5.1.1 Baseline model

We create a very simple baseline model. It is a function that takes in two words, and predicts rhyme if the substring from and including the last vowel is the same for both words (see implementation in Listing 2). This simple baseline gets an accuracy of 0.90 on the test set.

5.1.2 Model training and results

We train our models using the code from Minogue (2021) almost exactly as is. The only change we make is to add early stopping to the training loop, as to not waste energy by training the models unnecessarily long (in the original code the model is trained for 100 epochs). The early stopping callback monitors validation loss, and has a patience of 5 epochs.

[1]code here: https://github.com/minoguep/rhyme_detection

	Validation set accuracy
Model 1	0.94
Model 1.2	0.95
Model 2	0.92
Model 3	0.8
Model 3.2	0.7

Table 5.2: Accuracy on "Rhyme detection with Siamese LSTM" experiments

The first model, named model 1, is trained with the 1:1 ratio of positive to negative examples, exactly as in Minogue (2021). The model ends up with an accuracy of 0.94 on the validation set (see Table 5.2).

Model 1.2 is trained with a 2:3 ratio of positive to negative rhyme pairs, as in Haider and Kuhn (2018). We use the training set from model 1, but increase the number of negative rhyme pairs to achieve a 2:3 ratio. This model got an accuracy of 0.95 on the validation set, one percentage point better than model 1.

We also train a model with the dense rhyme pairs we collected as explained in chapter 4. Note that the number of unique words is the same, there are just more positive/negative rhyme relations between the same number of words. We use the 38 599 rhyme pairs extracted from the good buckets + manual clustering (selection 5 in Table 4.1). The negative rhyme pair selection is the same as with models 1 and 1.2 (selection 1 in Table 4.1). In this case, there are more positive examples than negative, so we use all 22 447 negative pairs, and sample the same number of positive rhyme pairs, for a total of 44 894 word pairs. As there is overlap with this set of rhyme pairs and the original, we make sure to remove all word pairs that are present in the validation set and test set. This gives us 41 176 word pairs to train the model. This model, which we call model 2, achieves an accuracy of 0.92 on the validation set, worse than both model 1 and model 1.2 .

Adding even more data, we use the rhyme pairs from the merged Wiktionary and manually clustered buckets (selection 7 in Table 4.1), and the negative word pairs from Wiktionary (selection 6 in Table 4.1). Model 3.1 is trained on a subset of 40 000 word pairs, and model 3.2 is trained on all 155 838 positive examples, plus the same number of negative examples, making a total of 311 676 examples. Both these models perform pretty bad, with a validation set accuracy of 0.81 for the smaller and 0.70 for the larger model. The training is ended after just 6-8 epochs, compared to between 20 and 80 epochs with the other models.

5.1.3 Faults with the model architecture

The siamese LSTM from Minogue (2021) is not commutative, that is model(A, B) != model(B, A). The reason for this is the Subtract-layer (see Figure 5.1). This leads to cases where the model gives a different rhyme prediction for a word pair depending on the order of the items in the pair.

Model	Val. set accuracy	Mirrored val. set accuracy
1	0.94	0.93
1.2	0.95	0.94
2	0.93	0.92
3	0.77	0.78
3.2	0.71	0.70

Table 5.3: SRN models' accuracy on mirrored validation sets

Commutative model	Accuracy
1	0.94
1.2	0.95
2	0.95
3	0.81
3.2	0.7

Table 5.4: Commutative SRN models' accuracy

We construct the mirrored version of the validation set S as follows:

$$S_{mirror} = \{\, (w_2, w_1) \mid (w_1, w_2) \in S \,\} \tag{5.1}$$

For model 1, the rhyme prediction differs between the original and mirrored versions for 38 of the 1448 pairs in the validation set. For model 1.2, this number is 36. For model 2, the predictions for 34 pairs are different when the pairs are mirrored. Model 3 gets different predictions depending on word order for 306 of the 1448 pairs in the validation set, and for model 3.1 this number is 363.

In Table 5.3 we see the impact on accuracy with the original and mirrored validation sets. For all models except model 3, the accuracy is 1 percentage point lower on the mirrored validation set. For model 3 the accuracy is 1 percentage point higher.

5.1.4 Improving the model architecture

We create a commutative model by adding a layer that performs element-wise absolute value after the Subtract-layer. The results of re-running the above experiments with the commutative versions of the models can be seen in Table 5.4. Commutative models 1 and 1.2 get the same accuracy as the original ones, which is an increase of 1 percentage point on the mirrored validation set. Commutative model 2 gets a 2 percentage point increase in accuracy compared to the original one. Commutative model 3 gets a 4 percentage point increase in accuracy compared to the non-commutative one. Model 3.1 accuracy decreases by 1 percentage point (Table 5.4). In general, the performance increased slightly when using a commutative model architecture.

Rhyme pair\model	baseline	1	1.2	2	3	3.2
sagt, sagd	**NR = Not rhyme**	**NR**	**NR**	**NR**	**NR**	R
vilje, vanilje	**R = Rhyme**	**R**	**R**	**R**	**R**	**R**
tid, svineri	NR	**R**	NR	**R**	**R**	**R**
vokser, bukser	R	R	R	R	R	R
ha det, badet	**R**	NR	NR	NR	**R**	**R**
se det, frede	NR	**R**	**R**	**R**	**R**	**R**

Table 5.5: Example word pairs and rhyme predictions from models (correct predictions in bold

5.1.5 Example predictions

We included some example word pairs, and the produced predictions from the commutative models in Table 5.5.

All models are able to correctly predict that *vilje* (vilje) 'will' and *vanilje* (vɑnilje) 'vanilla' rhyme. None of them are able to correctly predict not rhyme for the word pair *vokser* (vɔksɛɾ) 'grows' and *bukser* (bukseɾ) 'pants'. Only the largest model has an erroneous prediction for the word pair *sagt* (sɑkt) 'said' and *sagd* (sɑɡd) 'sawed'. The word pair *tid* (tiː/tiːd) 'time' and *svineri* (svineriː) 'mess, filth' rhymes if the speaker pronounces *tid* like (tiː). Interestingly, only model 1.2 did not predict rhyme for this word pair. We included two word pairs containing multiword rhymes, which is not something the models were trained at. We were positively surpised that all models got the pair *se det* (sɛdɛ) 'see it' and *frede* (frɛdɛ) 'protect'. The two largest models also got the word pair *ha det* (hɑdɛ) 'good bye, have it' and *badet* (bɑdɛ) 'the bathroom'.

5.1.6 Discussing and comparing the results

With the same model architecture, but only 1.4% of the data, the model trained on the NoRSC rhyme pairs gets a comparable accuracy, just 1 percentage point below Minogue. With the 2:3 ratio of positive to negative rhyme pairs, the model achieved the same accuracy as Minogue (2021). We realize that these results, having been obtained on two different data sets for two different languages, are not comparable. Even so, it is interesting to compare our results to these previously obtained using an identical architecture.

The non-commutative version of model 2, utilizing the dense rhyme pairs, gets worse results than both model 1 and 1.2, which were trained with much less data. The commutative version achieves the same accuracy as model 1.2, but we would have expected an increase in performance as the amount of data was doubled.

The models trained with the merged Wiktionary and dense pairs as the training data perform worse than the baseline. It seems that learning Wiktionary rhyme pairs does not improve the models ability to correctly predict the rhyme pairs in the validation set, rather, more data yield worse results.

We suspect the reason for this is that with the merged Wiktionary and dense pairs, new words are introduced. Though there is no overlap of rhyme pairs in the training sets for models 1, 1.2 and 2 and the validation set, the same words appear in different pair constellations across the sets. Ideally the models should learn something about which combinations of characters that create rhymes, and this should generalize to all words in Norwegian, both seen and unseen. It seems to be the case that the model having "seen" a word during training makes it more likely that it will correctly predict rhyme or not rhyme when it appears in a pair in the validation set. To confirm this, we create a separate test set, consisting of *only* the Wiktionay pairs (selection 6 in Table 4.1). Removing all pairs that appear in the training data of all the models, we now have a test consisting of words that are mostly unseen for models 1, 1.2 and 2. The baseline model achieves 0.93 accuracy on this set. On this test set, models 1, 1.2 and 2 all perform below the baseline. Models 3 and 3.2 achieve and accuracies of 0.96 and 0.98, respectively. See Appendix B for the full results.

It seems that our suspicions were correct. Models 3 and 3.2 have definitely learnt something, as we see both from the example pairs in Table 5.5 and the accuracy on the Wiktionary pairs. They simply do not perform well on the validation set.

We did not have the time to explore what the reason for this is in the work of this thesis, but it is definitely something that we would like to look into in the future. Having a third rhyme pair data set to use for comparison would be useful for that purpose.

5.1.7 Test set accuracy of the best model

Finally, the best models are model 1.2 and model 2 with their 0.95 accuracy on the validation set. They both score 0.96 on the test set.

5.2 Rhyme generation

Now that we have assessed the performance of an LSTM-based rhyme prediction model, we want to try to create a rhyme generation model. The idea is to train a model that takes one word as the input, and produces a set of words that rhymes with the input word. This can then be used to sample rhyming line-ending words in our poetry generation model, as in Van de Cruys (2020).

We utilize the buckets we extracted as described in chapter 4 for this. We design an LSTM-based model that takes a character-sequence as input, and outputs a bucket of words that rhyme with the input words. See Figure 5.2 for an overview of the model architecture. Thus, for each bucket, there are as many training examples as there are words in the bucket. The target is the bucket itself.

Recall that most of the good+manually clustered buckets are very small, with just 3 to 4 words per bucket (see Figure 4.7). All these 3- and 4-word buckets will only amount to one example from each bucket for each of the training, validation and test set. This naturally does not yield very good results. There is a trade-off between keeping as much data as possible, and not keeping buckets that are too small to learn from.

As discussed in chapter 4, when creating these buckets from the rhyme scheme annotations, there is no guarantee that all words that rhyme are connected and put in one bucket. This means we can expect that there are several

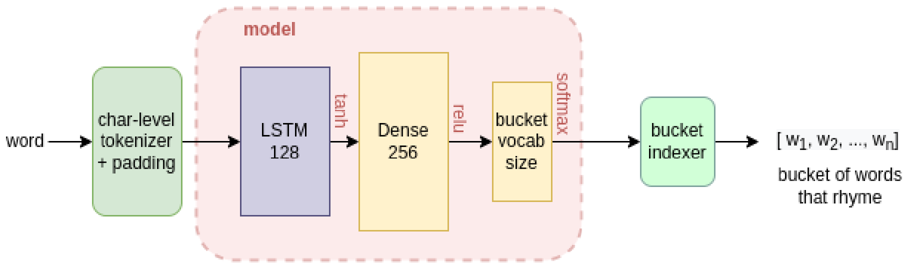

model
word
char-level tokenizer + padding
LSTM 128
tanh
Dense 256
relu
bucket vocab size
softmax
bucket indexer
[w_1, w_2, ..., w_n]
bucket of words that rhyme

Figure 5.2: Overview of the model architecture for the rhyme generation model

parallel buckets, buckets that represent the same rhyme-ending.

For the practical use of the model, this is not problematic. If the input word is “hest”, it does not matter much whether the model outputs the bucket {“fest”, “blest”, “krest”} or {“vest”, “pest”, “lest”}, as long as the output bucket contains rhyming words.

This will, however, affect model performance. It does not matter whether or not the words in the predicted bucket rhyme with the input word, the “wrong” bucket is the wrong bucket. In our experiments, we empirically assess the threshold for bucket size makes sense for a rhyme generation model.

5.2.1 Baseline model

Our baseline is a simple function. For an input word, it loops through the buckets, and the baseline rhyme-function (as described in Section 5.1.1) is used to see if the first word in the bucket rhymes with the input word. If they rhyme, this bucket is returned. If all buckets are looped though without the first word rhyming, a random bucket is returned.

5.2.2 Training models

As the different models trained have a different number of buckets, and thereby a different number of classes in the output layer, we can not use one test set across all models. We create training, validation and test sets for each set of buckets, and see how the number of buckets affects the models ability to correctly predict on its test set. The validation set is used for early stopping. All models are trained for 100 epochs, with a batch size of 64 and early stopping on validation loss with a patience of 5 epochs. The loss function is binary crossentropy, and the optimizer is adam. For each set of buckets used, the trained model is compared to how the baseline model performs on that set.

Dense buckets

The first set of buckets is the entire set of buckets from the good + manually clustered buckets (described in section 4.2.2). As mentioned above, most of these buckets contain only 3 or 4 words. The median bucket size is 5, and the average bucket size is 8.1.

In order to split this set into training, validation and test sets, we separate the 3- and 4-word buckets from the larger buckets. The words from these 274 buckets are split into train/validation/test tests with a ratio of 30/30/30, ensuring at least 1 example from each bucket in each set. The words from the rest of the buckets, 322 in total, are split into train/val/test sets with a 60/20/20 ratio. 279 of the 596 buckets are not present in the model's prediction on the test set. The models macro average F1 score is 0.39 on the test set. Comparably, the baseline on this set of buckets gets a macro average F1 score of 0.22.

The next set of buckets are the same as the one above, except that all buckets with fewer than 5 words are excluded. This nearly halves the number of buckets, as the total number of buckets is now 322. The median bucket size is 9, and the average bucket size is 12. The words from the buckets are split into train/validation/test sets with a 60/20/20 ratio, and the model is trained. This time, only 23 of 322 buckets are excluded from the test set prediction. This model gets a 0.76 macro average F1 score on the test set. The baseline on the other hand, achieves a macro average F1 score of 0.29 on this set of buckets.

We also use the same buckets as above, this time removing all with fewer than 10 words. This again halves the number of buckets, to 147. The median bucket size is 14.0, and the average bucket size is 18.7. The words from the buckets are split 70% to the training set, 15% to the validation set and 15% to the test set. This model gets a 0.95 macro average F1 score on the test set. Only one bucket is missing from the predictions. The baseline model achieves a macro average F1 score of 0.38,

Wiktionary buckets

The next set of buckets is the Wiktionary buckets (described in section 4.3). The Wiktionary buckets are very different from the good+manually clustered buckets. There are fewer buckets, and the buckets are much larger. After removing the 9 buckets that have less than 5 words, we are left with 18 buckets with an average size of 82.2, and a median size of 48.5 words per bucket.

The words from the buckets are split 70% to the training set, 15% to the validation set and 15% to the test set. The model trained on these data achieve a macro average F1 score of 0.91 on the test set predictions, and all buckets are present. The baseline achieves a macro average F1 score of 0.54.

Merged dense and Wiktionary buckets

With the merged buckets (described in section 4.4), we train three models, one with all buckets of size >=5, one with all buckets of size >=6, and one with all buckets of size >=10.

For the set of buckets of size 5 or more, the number of buckets is 331. The median bucket size is 9.0 and the average bucket size is 15.6. The data is split 60% to the train set and 20% each to the validation and test sets. The model trained on these buckets achieves a macro average F1 score of 0.91 on the test set. 13 buckets are missing from the test set predictions. The baseline function gets a macro average F1 score of 0.33.

Bucket-set modelled	Number of buckets	Number of unique words	Median bucket size	Macro average F1 score
Manual clusters	596	4818	5	0.39
Manual clusters size ≥ 5	322	3890	9	0.76
Manual clusters size ≥ 10	147	2751	14	0.95
Wiktionary size ≥ 5	18	1366	48.5	0.91
Merged size ≥ 5	331	5162	9	0.91
Merged size ≥ 6	283	4922	10	0.94
Merged size ≥ 10	161	4054	15	0.96

Table 5.6: Data set sizes and achieved macro average F1 scores for the different rhyme generation models

For the buckets of size 6 or more, the number of buckets is 283. The median bucket size is 10.0 and the average bucket size is 17.4. The data is split 70% to the train set and 15% each to the validation and test sets. The model got a 0.94 macro average F1-score on the test set, and 6 buckets were missing from the test set predictions. The baseline model got a 0.35 macro average F1-score on the test set.

For the buckets of size 10 or more, the number of buckets is 161. The median bucket size is 15.0, and the average bucket size is 25.2. The data is split 70% to the train set and 15% each to the validation and test sets. The model gets a 0.96 macro average F1 score on the test set, with no buckets missing from the test set predictions. The baseline model gets a macro average F1 score of 0.44 on this set of buckets.

5.2.3 Concluding remarks and examples

In general, we see an increase in performance as the smaller buckets are removed. All models outperformed the baseline model. The best set of buckets and the best model seems to be the merged Wiktionary and dense buckets set, with all buckets that contain less than 10 words removed. With this set of buckets, the model was able to cover all classes in the test set predictions, and the achieved macro average F1 score is the best across all models (see Table 5.6)

We tested some nonsensical words to see what other words they are predicted to rhyme with. See Table 5.7 for the predictions made by the best model. The

“Word”	Predicted rhyme bucket
gorlang	[’trang’, ’vingefang’, ’opprørstrang’, ’kjøkkengang’, ’sprang’]
knatt	[’vinter-natt’, ’betatt’, ’satt’, ’glatt’, ’fatt’]
pril	[’halofil’, ’bussfil’, ’vegetabil’, ’femmil’, ’postill’]
glunn	[’munn’, ’blomstergrunn’, ’offerlund’, ’barne-munn’, ’sekund’]
hjal	[’sjal’, ’kanal’, ’Kal’, ’ritual’, ’smal’]
plirk	[’mystikk’, ’leskedrikk’, ’stikk’, ’politikk’, ’forgikk’]
coll	[’avindsskjold’, ’gold’, ’protokoll’, ’kold’, ’vold’]

Table 5.7: List of some made up words and their assigned rhyme buckets

first five words of each predicted bucket is listed. All words except for “plirk” seem to have received a suitable word bucket.

Chapter 6

Poetry generation

Using the rhyme scheme annotated data set described in chapter 3, and the poetry generation model described in chapter 5, we here present an LSTM-based poetry generation model for Norwegian.

6.1 Model architecture

Our model is loosely based on Lau et al. (2018). We also use LSTMs for both text generation and rhyme modelling. Similarly to Lau et al. (2018), we sample rhyming line ending words that correspond to a rhyme scheme, and generate the rest of the stanza-based on those words. However, we do not explicitly model the stress or pentimeter in the source poetry. The only poetry feature that we explicitly model is rhyme, the rest we trust the language model to capture, if it is able to do so. And while Lau et al. (2018) is strictly focused on Shakespearean sonnets, with a strict rhyme scheme, our model can generate a stanza from any rhyme scheme.

The input for the poetry generation model is a rhyme scheme, and the output is a stanza with a rhyme pattern that matches the rhyme scheme. The model consists of a language model and a rhyme model, and the stanza is generated using both. We create two slightly different versions of the poetry generation model, depending on whether the language model is trained on complete stanzas or on individual lines from stanzas. The rhyme model is identical for both models, which is the rhyme generation model trained on the dense buckets of size $>=$ 10 (as described in Section 5.2.2 in the previous chapter).

In order to introduce the poetry generation model, we first describe the language model below.

6.1.1 Language model

Our language model is an LSTM-based language model for natural language generation. The model architecture is a sequential model with an embedding layer, an LSTM layer, and an output layer the size of the vocabulary (see Figure 6.1). For each time step during generation, the input is the word embedding of a word, and the output is the probability of each word in the vocabulary being the next word in the sequence.

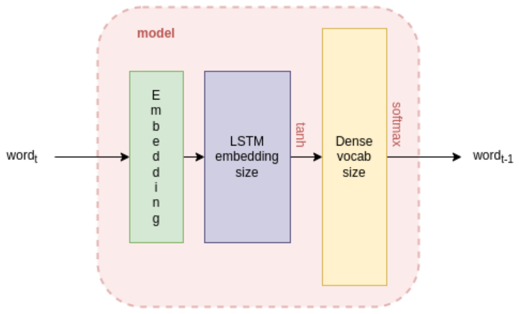


Figure 6.1: Language model architecture

X	y
<s>	jeg
<s>jeg	liker
<s>jeg liker	fisk
<s>jeg liker fisk	</s>

Table 6.1: Teacher forcing training data for the sentence *jeg liker fisk* 'i like fish'

The model is trained using *teacher forcing*; each instance in the training data is a sequence of words, and the target is the next word in the sequence. Thus, for the sentence *jeg liker fisk* 'i like fish', with teacher forcing the training data would be as in Table 6.1. '<s>' and '</s>' are special tokens to mark sequence start and sequence end, respectively.

For a model trained with teacher forcing in this manner, simply inputting the sequence start token starts generation. If the next predicted token is the sequence end token, generation stops and the full generated sequence is returned.

As we already mentioned, we train two different language models, a line-based one and a stanza-based one. For the line-based model, each stanza is split on line breaks, and training data is constructed for each line. That is, the sequence start and sequence end markers are placed on either side of each line, and training data is constructed as in Table 6.1.

For the stanza-based model, the start and end of sequence tokens are placed on either size of each stanza. Here we introduce another special token, '<n>' representing the line break. The idea is that this way, the model will produce stanzas with a consistent theme throughout the stanza. In contrast to this, the line-based model can only access the context of each line. To create a stanza with the line-based model, one has to generate the number of lines wanted in the stanza separately, which can lead to the content of the lines being completely unrelated to each other.

An important detail is that our models are trained on reversed data. Instead of predicting the next word, they predict the previous word. This way, we can enforce rhyme by first deciding rhyming line ending words, and generate the rest of the sequences backwards.

```
on input rhyme_scheme:
    for each unique symbol in rhyme_scheme:
        language_model.generate_line_ending_word().

    for each line ending word w:
        rhyme_model.get_bucket(w)

    for each symbol in rhyme_scheme:
        sample a line_ending word from the correct bucket

    # Now there is one line-ending-word for each line, and
    # rhyme relations are according to rhyme_scheme

    for each line ending word w:
        generate_line(w)
        (and reverse line)

    return the lines
```

Listing 3: Simplified pseudocode for line-based stanza generation

Language model training

We reason that for this task, overfitting on the training data is not a problem, as we want the model to learn the source corpus and produce similar texts. This is not a model that needs to be able to generalize and be applied to unseen texts. We therefore train both language models on the entire corpus data, without splitting it into training and test sets. Both models are trained for 100 epochs, with a batch size of 256. The optimizer is adam, and the loss function is sparse categorical crossentropy.

6.1.2 Generating rhyming stanzas

Now that we have described how the language models work, we can go on to describe the full poetry generation model. Recall from chapter 5 that the rhyme generation model returns a *bucket* of rhyming words for any word input.

For the line-based poetry generation model, the basic stanza generation algorithm works as seen in Listing 3. The poetry generation model takes in a rhyme scheme, and using the rhyme scheme, generates the appropriate number of rhyming line-ending words using the language and rhyme generation models. Then, each line is generated backwards, the prompt being the end of sequence marker and the line-ending word. The generated lines are stacked together and the result is a stanza following the provided rhyme scheme.

The stanza-based approach is a slightly more complicated. Here the method for sampling from the language model is modified so that the sequence is returned when the line break token is predicted. The next line ending word is inserted into the sequence, and it is sent back to the language model to continue

```
on input rhyme_scheme:
    for each unique symbol in rhyme_scheme:
        language_model.generate_line_ending_word().

    for each line ending word w:
        rhyme_model.get_bucket(w)

    for each symbol in rhyme_scheme:
        sample a line_ending word from the correct bucket

    # Now there is one line-ending-word for each line, and
    # rhyme relations are according to rhyme_scheme

    stanza_so_far = ""
    for each line ending word w:
        stanza_so_far += w
        g = language_model.generate_line(stanza_so_far)
        stanza_so_far += g

    return stanza_so_far
```

Listing 4: Simplified pseudocode for stanza-based stanza generation

generation. In this way, the rhyme scheme is ensured, and the context of the whole stanza (so far) is available to the language generation model. See listing 4 for pseudocode for the generation function using the stanza-based language model. The full code for both models can be found on the master thesis github[1]

A parameter to the poetry generation algorithm that is not shown in the simplified algorithms above is the temperature-parameter. In the language model, the default is to choose the most likely next word for a given input word during generation. With temperature, the output probabilities are changed slightly using a multinomial probability distribution. This ensures higher exploration, as the choice of the next word is no longer completely deterministic. The higher the temperature, the more exploration. For our poetry generation models, the stanzas are produced using a temperature of 0.5. We decided on this value after experimenting with a couple of different values.

[1]https://github.com/titaenstad/mester

6.2 Evaluation

As stated in the introduction, our modelling objectives are to investigate whether we can create a Norwegian poetry generation model that produces poetry of a quality such that

1. it cannot be discerned from poetry written by humans and
2. it rhymes.

6.2.1 Approach

As in the previous research explored in chapter 2, we also make use of human evaluation in order to evaluate the generated poetry. We use both side-by-side and standalone evaluation. In side-by-side evaluation, the evaluator is presented with two stanzas, and asked to pick out the one that is written by a human. In standalone evaluation, the evaluator is presented with one stanza at a time, and is asked whether or not they believe it is written by a human.

The evaluators are further asked to rate how well each stanza rhymes, on a scale of 0 to 3. 0 means that the stanza contains no rhyme. 1 means that the stanza contains some almost-rhyme. 2 means that the stanza has some rhyme, for example that two lines of the stanza rhyme, but the rest do not. 3 is used when is it obvious that the stanza was written to rhyme. See Appendix C for the full evaluation instructions.

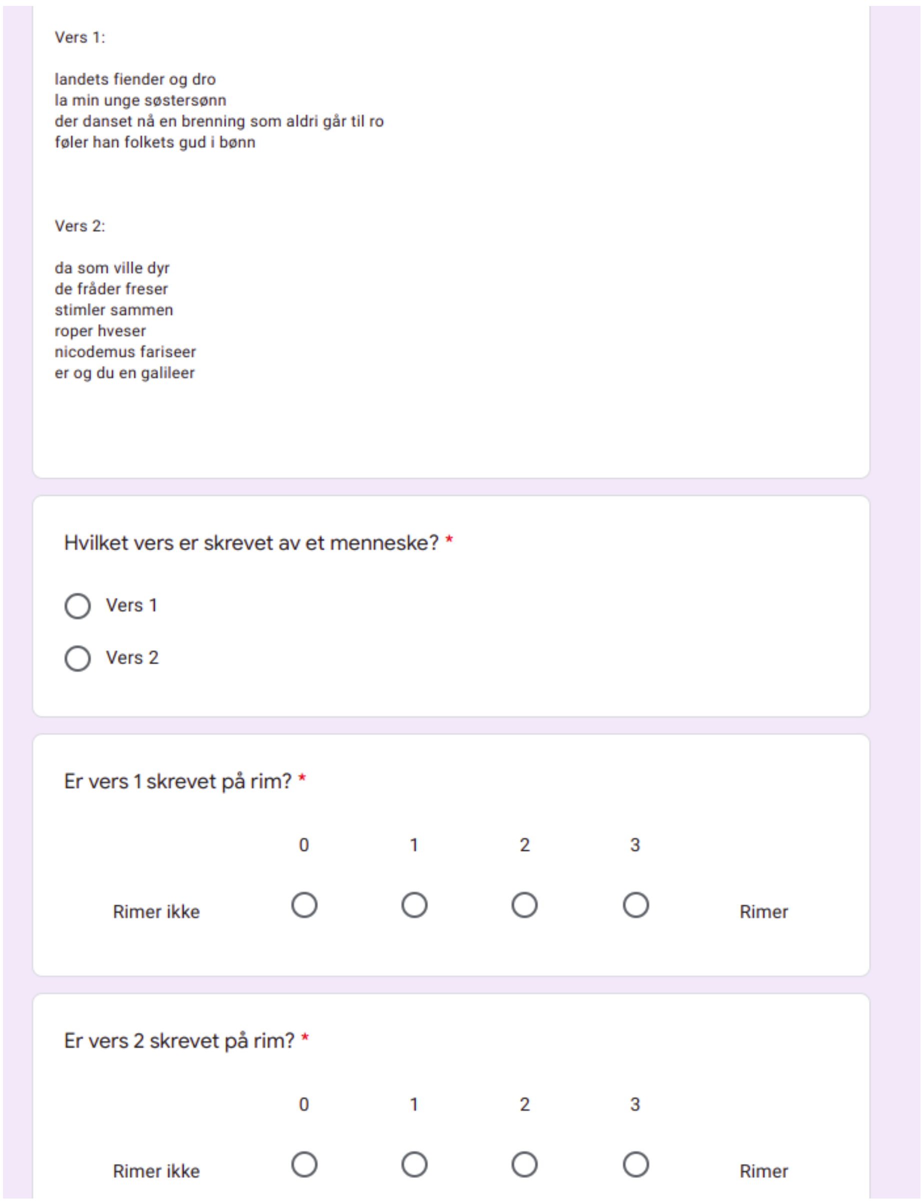
We create 4 web forms, to minimize the work load of the evaluation, and this way get more people to participate. We generate 40 stanzas with each model, 10 stanzas for each form. We use the top 10 most frequent rhyme schemes from NoRSC, and generate 4 stanzas for each scheme. The models are the baseline model (the stanza-based language model with no rhyme component), the line-level poetry generation model and the stanza-level poetry generation model. 5 stanzas from each model are used for side-by-side evaluation, and 5 stanzas are used for standalone evaluation.

We randomly sample stanzas from the original data set to fill in the human produced stanzas for the side-by-side evaluation, plus 5 stanzas for the standalone evaluation. The stanzas from the original data set are tokenized in the same way as the input data for the poetry generation model, so they look similar. In total, each form has 15 pairs of stanzas for side-by-side evaluation, and 20 single stanzas for standalone evaluation. In total, 120 generated stanzas, and 80 stanzas from the original NoRSC data set are evaluated by humans (see Table 6.2).

The forms were posted on social media, and were open for 3 days before they were closed. See Figure 6.2 for an example of how the evaluation interface looked.

6.2.2 Results

Each form received between 6 and 7 answers. In total, 27 answers to the forms were submitted. In the following, we examine the results, aiming to answer some of our research questions.


Vers 1:

landets fiender og dro
la min unge søstersønn
der danset nå en brenning som aldri går til ro
føler han folkets gud i bønn

Vers 2:

da som ville dyr
de fråder freser
stimler sammen
roper hveser
nicodemus fariseer
er og du en galileer

Hvilket vers er skrevet av et menneske? *

○ Vers 1

○ Vers 2

Er vers 1 skrevet på rim? *

	0	1	2	3	
Rimer ikke	○	○	○	○	Rimer

Er vers 2 skrevet på rim? *

	0	1	2	3	
Rimer ikke	○	○	○	○	Rimer

Figure 6.2: Evaluation interface from side-by-side evaluation

	Baseline model	Line-based model	Stanza-based model	NoRSC stanzas
Side-by-side number of stanzas	20	20	20	60
Stand-alone number of stanzas	20	20	20	20
Total	40	40	40	80

Table 6.2: Number of stanzas evaluated per model

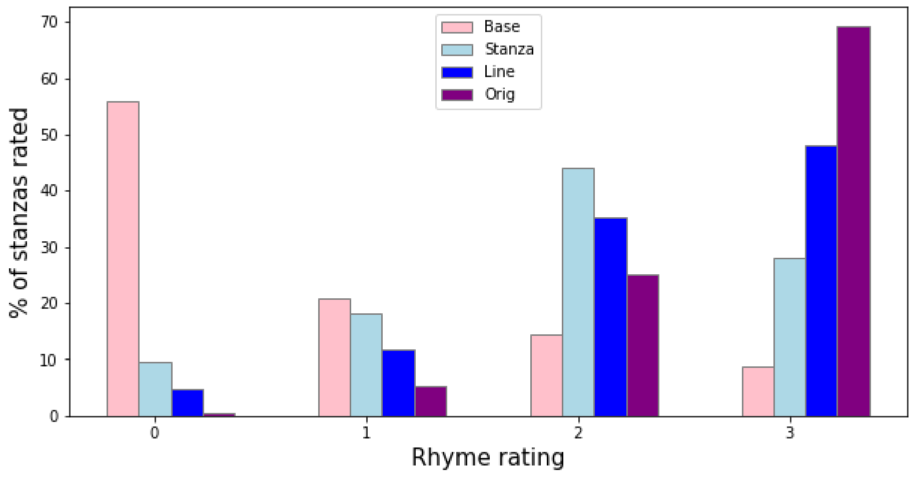


Figure 6.3: Frequency of rhyme rating for the stanzas

Majority score	Baseline model	Stanza-based model	Line-based model	NoRSC stanzas
<1	27	3	1	0
<2	34	9	1	1
<3	38	29	17	17
Total	40	40	40	80

Table 6.3: Majority rhyme score for the different stanza types

6.2.3 Rhyme ratings

We find that the stanzas that got the best overall rhyme rating are the original NoRSC stanzas. They got an average rhyme rating of 2.63. The second best stanzas are those generated by the line-based model, with an average rhyme rating of 2.27. For these stanzas, 3 was the most frequent rating (see Figure 6.3). The stanzas produced by the stanza-based model got an average rhyme rating of 1.91, and were most frequently rated 2. The baseline model, which was simply the stanza-based model without the rhyme-component, produced the least rhyming stanzas, with an average rating of 0.76, and 0 being the most frequent rhyme rating.

From these data we can conclude that both the line-based and stanza-based poetry generation models were able to model rhyme better than the baseline model, but not as well as the original data. The line-based poetry generation performed significantly better than the stanza-based poetry generation model, which is surprising, as their rhyme model is identical.

Instead of averaging the rhyme scores, another way to interpret the annotations is by choosing the majority score for each stanza. Applying this to the stanzas produced by the line-based model, we get that only 1 stanza out of 40 has a majority rhyme score that is less than 2. 17 of the stanzas have a majority

Stanzas	% of times rated to be written by a human in side-by-side evaluation	% of times rated to be written by a human in stand-alone evaluation	% of times rated to be written by a human in total
NoRSC stanzas	85.19	82.96	84.63
Baseline model	19.26	25.19	22.22
Line-based model	17.78	48.89	33.33
Stanza-based model	7.41	11.11	9.26

Table 6.4: Frequency of 'written by a human'-ratings for the different types of stanzas in side-by-side and standalone evaluation

rhyme score less than 3 (see Table 6.3). These numbers are the same for the original stanzas, but as there are twice as many of these, this means that the line-based model is about half as good at rhyming as the authors of the original stanzas.

For the stanza-based model, there are 9 stanzas with a majority score less than 2, and 29 stanzas with a majority score less than 3. For the baseline model, 34 of 40 stanzas have a majority score less than 3 (Table 6.3.

We reiterate our research question RQ2: How consistent are our poetry generation models at generating rhyming poetry? And answer it as follows: The line-based poetry generation model is fairly consistent at generating rhyming poetry. The baseline model is very consistent at generating poetry that does **not** rhyme. The stanza-based poetry generation model is not very consistent.

6.2.4 'Written by a human'-ratings

In Table 6.4 we see the frequency with which the different stanzas have been rated to be 'written by a human'. Across all models, the generated poetry is more often rated 'written by a human' in the standalone evaluation than in the side-by-side. This was expected, and it is also seen in Nikolov et al. (2020).

In the side-by-side evaluation, the original NoRSC stanzas were rated 'written by a human' 85.19% of the time (see Table 6.4). This means the evaluators combined got an accuracy of 85.19%, and that they in 14.81% of the cases annotated a generated stanza as written by human.

In the side-by-side evaluation, model that most often was rated 'written by a human' is the baseline model, as 19.26% of the ratings for the stanzas generated by the baseline model were 'written by a human'. The stanzas generated by the line-based model were rated to be written by a human 17.78% of the time, and the stanzas from the stanza-based model 7.41% of the time (see Table 6.4).

In the standalone evaluation, the original NoRSC stanzas were rated 'written by a human' in 82.96% of the ratings. This is a slight decrease from the side-by-side evaluation. This can indicate that it is easier to decide if a stanza is “real” if it is presented in a pair.

From the standalone evaluation, the best model is the line-based model. The stanzas generated by the line-based model are rated 'written by a human' in 48.89% of the ratings. The next best, with regards to being rated as written by a human, is the baseline model. Its stanzas were rated to be written by a human 25.19% of the time. The stanzas generated by the stanza-based model were rated to be written by a human in 11.11% of the ratings.

It is evident that the none of the models were able to perfectly mimic poetry written by humans. The best results were obtained by the line-based model in standalone evaluation, but even here, the evaluators were fooled less than half of the time.

If we compare our results from side-by-side evaluation to Nikolov et al. (2020), where the evaluators only mislabelled 7% of the texts, our results look good. However, if we compare our results to Lau et al. (2018) or Jhamtani et al. (2019), where the human evaluators mislabelled 47% or 44%, respectively, our results do not look that good any more. One major difference here is that the evaluators in Nikolov et al. (2020) were experts on the genre, which we can expect increases their ability to discern real lyrics from 'fake'.

Addressing research question RQ3: To what degree is our generated poetry believed to be written by a human? We answer: to some degree. We conclude that we were able to achieve one of two modelling objectives. We were able to create a poetry generation that rhymes, but the poetry is not indiscernible from poetry written by humans.

6.2.5 The relation between rhyme and perceived “written by a human”-ness

In this section we address research question RQ4: Is there a connection between the generated poetry rhyming and it being perceived to be written by a human ?

The stanza-based model scores higher than the baseline model when it comes to rhyme, but worse when it comes to being rated as written by a human. Therefore, we cannot conclude that there is a direct connection on model level. However, if we look at the ratings for each type of stanza, we see another picture.

In Figure 6.4 we see the proportion of stanzas being rated written by a human/being rated not written by a human for each rhyme score for the baseline model. We see a clear, linear pattern: the higher the rhyme score, the more are the stanzas rated 'written by a human'. We see the same pattern for the line-based model in Figure 6.5.

For the original stanzas, we observe that fewer stanzas are rated to be written by a human when the rhyme score is 0 (see Figure 6.7). The other three scores seem to be relatively similar, though, with around 80% of the stanzas being rated 'written by a human'.

For the stanza based model, we see that the stanzas with rhyme scores 2 and 3 are more frequently annotated to be written by a human than those that are scored 0 and 1. However, it looks like the score that coincides with the most “written by a human”-ratings is 2, not 3.

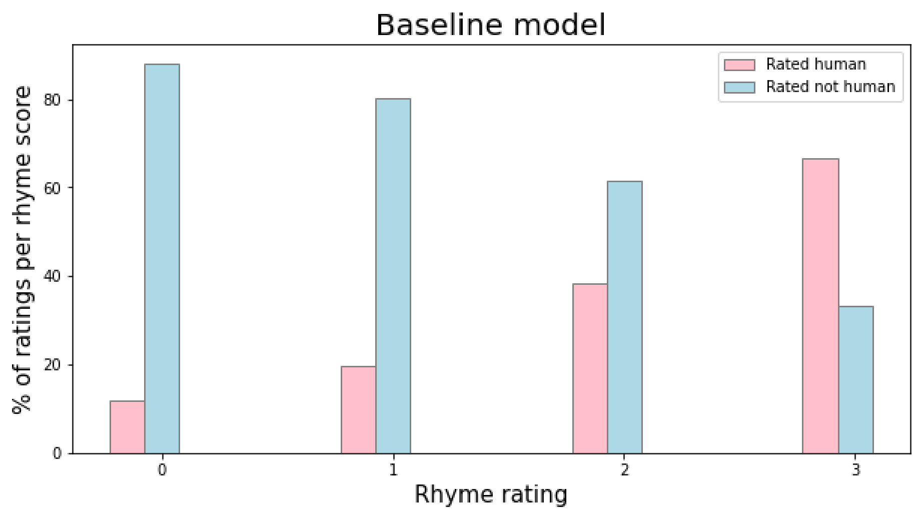


Figure 6.4: Proportion of stanzas rated 'written by a human' for each rhyme score for baseline model

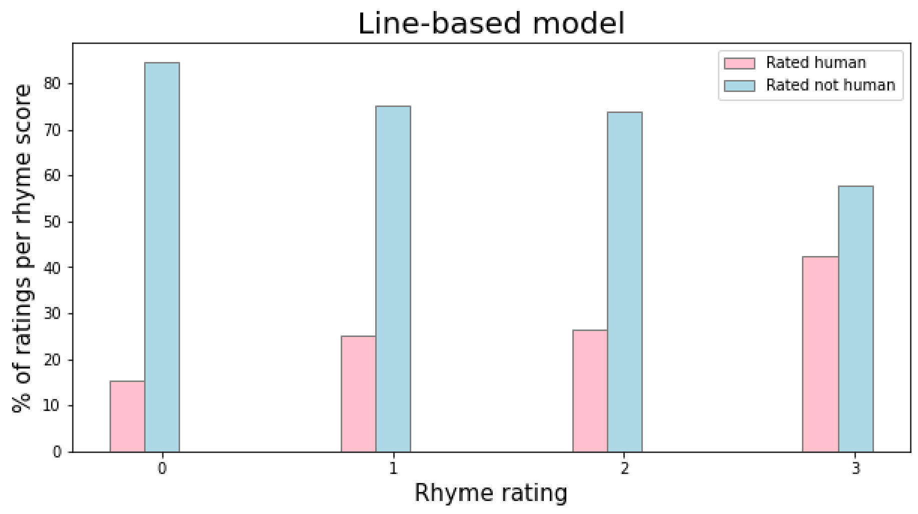


Figure 6.5: Proportion of stanzas rated 'written by a human' for each rhyme score for line-based model

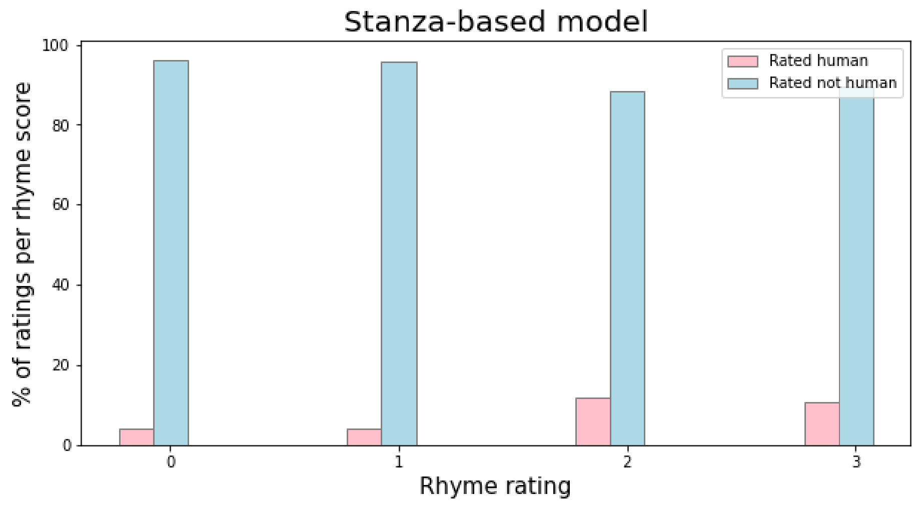


Figure 6.6: Proportion of stanzas rated 'written by a human' for each rhyme score for stanza-based model

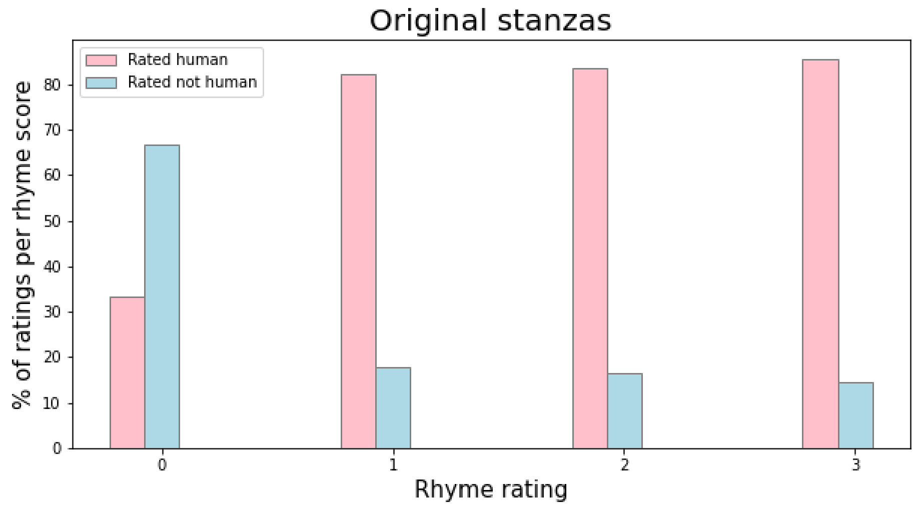


Figure 6.7: Proportion of stanzas rated 'written by a human' for each rhyme score for original stanzas

6.2.6 Discussing the stanza-based model

The stanza-based model generally performs worse than expected. It is surprising that the rhyme scores for the stanza-based model are that much lower than the line-based model, considering that they use the same rhyme model.

One possible reason for this is a the way the stanza-based poetry generation model works. Recall section 6.1.2 where we describe the generation algorithm for the stanza-based model. The line-ending words are sampled from the rhyme model. These are inserted into the sequence during generation, which are then sent back to into the language model. Because the line-ending words are selected solely based on the rhyme model's probabilities, this can lead to word sequences that are very unlikely according to the language model.

But this cannot be the only reason. In example 6.1 and 6.2, we see two stanzas. In both, the last line looks highly ungrammatical, with too many words bunched together with little to no grammar. But since the stanzas are generated backwards, we cannot blame this on unlikely words being inserted. When these lines were generated, no words (except for the very last, the prompt that started generation) had been inserted. We do not know why this happens.

(6.1) fjellene i storfangst går og fuglesang
hvis morgenglans er klokkeklang
lukk mitt nordlys jøde kastes begynte stamsund flod nærmer sang

(6.2) russlands sære kogleri
jeg kom mine mål til å eie
og det er en gyllen dag så bred
han føler tåker og av deres kunster
for norges liv
norske norske norske norske norske hunger grønnes varslet mumler urørt
urørt favntak betenk bitterhet slemt karpus grisjka løvspring bakved 1942

Another thing that is interesting about 6.1, is that though all lines in the stanza seems to rhyme, it has consistently been rated 2 more often than 3. It might be the case that if the structure of the poetry is bad, they are not perceived to rhyme as well. We need to make further experiments before we can conclude any of these suggestions.

6.2.7 Controversial stanzas

As mentioned, each form received 6-7 answers, meaning that every stanza has been evaluated by 6-7 different people. Some stanzas have very disagreeing ratings. For example, there are 25 stanzas where there are at least 3 people that annotated the stanza to be written to be human, *and* at least 3 people that annotated the stanza to *not* be written by a human. Below are some of these stanzas:

(6.3) med yndig majestet han går
som gjorde sinnet blått av romantikk
vansiret av laster sykdom og sår
plutselig stod stauper igjen for mitt blikk

(6.4) skurken saulus løfter armen
atter dette stikk i barmen
denne angst de syner røde
taus han stirrer på den døde

(6.5) nuets underveis
se det er det jovisst
der ligger så stille mot vest hvor sol går ned
som klippene og sluttet kvist

(6.6) så må de ha vise
kan trenge seg inn
nå er den å farvel farvel
å farvel farvel
du som fulgte i somre atten

6.3 was generated by the line-based poetry generation model. 6.4 is from the original data set. 6.5 was generated by the stanza-based poetry generation model, and 6.6 was generated by the baseline model.

As for controversial stanzas with regards to rhyme, there are 19 stanzas that have been given both a score of 0 and a score of 3 in rhyme rating. We list some of these below:

(6.7) hvor bårer med skum over utsteinen slår
og kjølens siste bakker i det kildested
det første hender med høstens redsler død
hin yngste apall strør blikke hevet goder tå flyver ned

(6.8) da lyner et lys ned i dypet
en gnistrende sprøyt av ild
det glitrer av blånende bølger
er havet likevel til

(6.9) inn i fjellets flammesky
en lovsangs brus som storm mot sky
fra sinnet på ny og på ny
fra sinnet på ny og på ny

(6.10) og tross den tross dødninghaven
over vennens liv er bange
hver gang han synger se ham
den er seirens dag til gry

6.7 was generated by the stanza-based poetry generation model. It received the rhyme scores [3, 2, 2, 2, 2, 1, 0]. 6.8 is from the NoRSC data set. It received the rhyme scores [3, 3, 3, 2, 2, 2, 0]. 6.9 was generated by the line-based poetry generation model, and received rhyme scores of [3, 3, 3, 3, 3, 3, 0]. 6.10 was generated by the baseline model, and received rhyme scores of [3, 1, 1, 0, 0, 0, 0].

It seems that for most of the controversial stanzas with regard to rhyme score, there is only one score of 3 or 0, and the rest of the rhyme scores are more or less in agreement. This is the case for examples 6.8, 6.9 and 6.10. In these cases, the cause for the score might have been a mis-click made by one evaluator. Example 6.7, on the other hand, received all four possible rhyme scores, so it is not as easy to just dismiss this as mis-clicks. Still, the most frequent score for this stanza, 2, is the appropriate annotation according to the instructions.

6.2.8 Original stanzas rated 'not written by a human'

There were three stanzas from the NoRSC data set that received more 'not written by a human' ratings than 'written by a human' ones. All these three were from the side-by-side evaluation, which means that it is not just the quality of the original stanza, but also the quality of the stanza it was paired with, that contributes to the ratings.

The stanza in 6.11 was annotated to be written by a human 2 times, and not written by a human 5 times. The stanza it was paired with, 6.12, was generated by the line-based poetry generation model. Interestingly, though 6.11 is follows a solid ABAB rhyme pattern, it received the rhyme ratings [3, 3, 2, 2, 2, 1, 0]. 6.12 also follows the same rhyme pattern, and this was rated slightly higher with the rhyme scores [3, 3, 2, 2, 2, 1, 1].

(6.11) tillykke med dåden dere frelste imperiet
mac donald og thomas og henderson
og jobben var hård dere fortjener en ferie
men først må dere kysse hr. baldwins hånd

(6.12) han er jo
hva var det en guds fiolin
templet du har skjendet med vold og blod
især i det siste de trenger seg inn

The stanza in 6.13 was annotated written by a human 3 times, and not written by a human 4 times. It was paired with 6.14, which was generated by the line-based poetry generation model. The original stanza received perfect rhyme scores from all evaluators.

(6.13) det følte hver som kom
fra reis igjen og så seg om
det følte hver som gikk
i siste avskjedsblikk

(6.14) sjelevingens hvilegren
har båret det ord og det ble igjen
frem stormer egyptens armé
et dryssende stjernegry
fra sinnet på ny og på ny
hans veier ble lys for lys og fred

The stanza in 6.15 was annotated written by a human 3 times, and not written by a human 4 times. It was paired with 6.16, which was generated by the baseline poetry generation model. This example also demonstrates that a stanza rhyming does not make seem more like it is written by a person, as 6.15 does not rhyme at all. It makes up for the lack of rhyme by being surprisingly coherent and grammatical.

(6.15) å våke over fedrelandets lover
å sørge for at ingenting forlises
er smukt betryggende for folkehellet
men det er skjønnere med ånd å våge
å slå med moses tryllestav på fjellet
skjønt nytten ei kan fattes og bevises

(6.16) men se han kommer med venner
på klippens bryst da lød
evangeliets herlige bud

Chapter 7

Conclusion

7.1 Summary

Chapter 1: Introduction

In this chapter, we briefly presented the fields of poetry generation and rhyme modelling. We described our motivation, which was to fill an obvious gap in the field: that no prior research has been done with Norwegian data.

We also presented our main modelling objective, which is to investigate whether we can create a Norwegian poetry generation model that produces poetry of a quality such that

1. it cannot be discerned from poetry written by humans and
2. it rhymes.

We presented our four **research questions**:

RQ1: Is there a strong enough relation between written Norwegian and spoken Norwegian to accurately model rhyme based on text data?

RQ2: How consistent are our poetry generation models at generating rhyming poetry?

RQ3: To what degree is our generated poetry believed to be written by a human?

RQ4: Is there a connection between the generated poetry rhyming and it being perceived to be written by a human ?

Chapter 2: Background

In this chapter, we discussed poetry and rhyme, poetry generation and rhyme modelling. We explored previous work related to our modelling objectives, focusing on data sets, model architectures and evaluation methods. We saw that various model architectures could be used for poetry generation. All papers used human evaluation and some variant of the Turing test to evaluate whether the generated poetry was perceived to be written by a human.

Chapter 3: Creating a corpus of rhyme scheme annotated Norwegian poetry

In this chapter, we presented NoRSC: Norwegian Rhyme Scheme Corpus, a data set of rhyme scheme annotated Norwegian poetry, as well as the work that went into creating this data set. The final data set consists of 5158 stanzas, or a total of 26 198 lines of poetry. 100 stanzas were doubly annotated, and the inter-annotator agreement was shown to be high, with a Cohen's kappa of 0.96.

Chapter 4: Rhyme pair collection

In this chapter, we described how to extract rhyme pairs from rhyme scheme annotated poetry. We also discovered a graph based approach to harvest additional rhyme pairs from the same set of stanzas, so called *dense rhyme pairs*.

Chapter 5: Rhyme modelling

In this chapter, we used the rhyme pairs from the previous chapter to train models for two different rhyme modelling tasks: rhyme detection and rhyme generation. For both tasks, the input words were represented as character vectors. The rhyme detection model is a Siamese LSTM that takes in two words and predict whether they rhyme. The rhyme generation model is an LSTM-based model that takes in a word and outputs a set of words that rhyme with the input word.

The best models for both tasks received high scores, an accuracy of 0.95 for the best rhyme detection model, and a macro average F1 score of 0.96 for the best rhyme generation model. The results from the best rhyme detection model are comparable to previous work with English and German data. We answered **RQ1**: there is a strong enough relation between written Norwegian and spoken Norwegian to accurately model rhyme based on text data.

Chapter 6: Poetry generation

In this chapter, we train our poetry generation models and evaluate them using human evaluation. The poetry generation model consists of an LSTM-based language model for text generation, and a rhyme generation model from the previous chapter. We train two different versions of the language model, a line-based variant and a stanza-based variant. During generation, line-ending words are sampled from the rhyme generation model, and the rest of the stanza is generated with the language model. The baseline poetry generation model is simply the stanza based language model, with no rhyme model to enforce rhyme.

40 stanzas are sampled from each of the three models, plus 80 from the original source data set. Half of the generated stanzas are evaluated side-by-side with a stanza from the source data set. The other half is evaluated standing alone.

The stanzas are evaluated by volunteers, and each stanza is evaluated by 6-7 different people. For each stanza, the evaluators are asked to rate the quality of the rhyme on a 0-3 scale, and also if they think it is written by a human.

Our best model, the line-based model, shows a fairly good performance at producing rhyming poetry, receiving an average rhyme score of 2.27. In comparison, the stanzas from the source data set received an average rhyme score of 2.63. The stanzas generated by the stanza-based model received mediocre to low rhyme scores, with an average rhyme score of 1.91. The stanzas generated by the baseline model received an average rhyme score of 0.76.

From the results of the human evaluation, we answered **RQ2**: our best model is fairly consistent at producing rhyming poetry. Our baseline model is consistent at producing poetry that **does not** rhyme. Our stanza-based model is not very consistent.

We answered **RQ3**: The results from the human evaluation show that none of our models are able to perfectly mimic real poetry. For the standalone evaluation, the best performing model with regards to being perceived as 'written by a human' is the line-based model. The stanzas generated by the line-based model were rated 'written by a human' by 48.9% of the evaluators.

Somewhat surprisingly, the baseline model performs better than the other two in the side-by-side evaluation. When a stanza produced by the baseline model was paired with a stanza from the original NoRSC data set, 19% picked the generated stanza to be written by a human. The stanza-based model performs the worst, fooling the evaluators less than 10% of the time across both side-by-side and standalone evaluation.

Finally, we answered **RQ4**: there is *some* connection between the generated poetry rhyming and it being perceived to be written by a human. On model level, we could not conclude that there was a connection, as the worst model on the Turing-like test was the second best with regards to rhyme. However, examining the results for each model separately, we saw an obvious tendency. For the stanzas produced by the baseline model, the stanzas produced by the line-based model, and the stanzas from the original data set, there was a correlation between high rhyme score and stanzas being annotated as written by a human. For the stanza-based model, the stanzas with the rhyme score 2 were most often perceived to be written by a human.

7.2 Contributions, limitations and future work

In the work of this thesis, we created the first publicly available data set of rhyme scheme annotated Norwegian poetry. We are also, to our knowledge, the first to experiment with using Norwegian data in the field of poetry generation. Hopefully, this contribution will enable more people to attempt to model Norwegian poetry.

For future work, we would like to continue the rhyme scheme annotation to expand the NoRSC data set. We are especially interested in including more nynorsk material. In this case, we would try to semi-automate the text preprocessing, as doing it manually was a bit too time consuming. This, as well as collecting more, newer data would be a big contribution to the field.

In both the rhyme modelling chapter and the poetry generation chapter, we got some results which we found difficult to explain. The fact that the rhyme detection models trained on the NoRSC rhyme pairs did not generalize well to the Wiktionary pairs is definitely something which we would want to find the reason for. Also, we could not fully explain with certainty why the stanza-based poetry generation model produced stanzas of such low quality compared to the line-based and baseline poetry generation models.

With the time limit and scope of this thesis, we did not have time to delve that deep into exploring different models, or even spend extended time on hyper-parameter optimization and experiments with the models we did use. It would have been interesting to experiment with all the different model architectures we described in the backgrounds chapter. This work serves as a first experiment of poetry generation for Norwegian, a baseline that can be improved upon in future work.

For future work, it would be interesting to include phonetic information from *NLB Pronunciation Lexicon for Norwegian Bokmål*[1] or other resources could lift the quality of the rhyme modelling further.

Recent work on poetry and lyric generation make use of pretrained transformer models such as T5 (Tian and Peng, 2022; Ram et al., 2021), BART/m-BART (Ivanova and Uotila, 2021) and GPT (Liao et al., 2019). There has not been any Norwegian versions of these models publicly available, until a Norwegian GTP-J was released just two weeks before this thesis was submitted[2]. Utilizing this model for poetry generation has the potential to greatly improve the results with regard to the 'written by a human'-ratings.

[1]https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-52/
[2]https://huggingface.co/NbAiLab/nb-gpt-j-6B

Bibliography

Addanki, Karteek and Dekai Wu (2013). “Unsupervised rhyme scheme identification in hip hop lyrics using hidden Markov models.” In: *International conference on statistical language and speech processing*. Springer, pp. 39–50.

“Åndsverkloven” (2018). In: URL: https://lovdata.no/lov/2018-06-15-40/%C2%A711 (visited on 11/03/2021).

Bamman, David, Brendan O’Connor, and Noah A. Smith (Aug. 2013). “Learning Latent Personas of Film Characters.” In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics: Sofia, Bulgaria, pp. 352–361. URL: https://www.aclweb.org/anthology/P13-1035.

Berge, Bjørn Jonssønn (2016). “Rimets aktualitet. Rimets funksjoner i nyere norsk lyrikk.” MA thesis.

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL].

Bull, Tove (2022). *Ivaar Asen i Store norske leksikon på snl.no*. URL: https://snl.no/Ivar_Aasen (visited on 03/24/2022).

Devlin, Jacob et al. (June 2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics: Minneapolis, Minnesota, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

Gatt, Albert and Emiel Krahmer (2018). “Survey of the state of the art in natural language generation: Core tasks, applications and evaluation.” In: *Journal of Artificial Intelligence Research* 61, pp. 65–170.

Gillioz, Anthony et al. (2020). “Overview of the Transformer-based Models for NLP Tasks.” In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pp. 179–183. DOI: 10.15439/2020F20.

Greene, Erica, Tugba Bodrumlu, and Kevin Knight (Oct. 2010). “Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation.” In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics: Cambridge, MA, pp. 524–533. URL: https://www.aclweb.org/anthology/D10-1051.

Gundersen, Dag (2022). *Samnorsk i Store norske leksikon på snl.no*. URL: https://snl.no/samnorsk (visited on 03/28/2022).

Haider, Thomas and Jonas Kuhn (Aug. 2018). "Supervised Rhyme Detection with Siamese Recurrent Networks." In: *Proceedings of the Second Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*. Association for Computational Linguistics: Santa Fe, New Mexico, pp. 81–86. URL: https://www.aclweb.org/anthology/W18-4509.

Hämäläinen, Mika and Khalid Alnajjar (2019). "Let's FACE it. Finnish Poetry Generation with Aesthetics and Framing." In: *arXiv preprint arXiv:1910.13946*.

Hämäläinen, Mika and Khalid Alnajjar (2021). *The Great Misalignment Problem in Human Evaluation of NLP Methods*. arXiv: 2104.05361 [cs.CL].

Hartuv, Erez and Ron Shamir (2000). "A clustering algorithm based on graph connectivity." In: *Information Processing Letters* 76(4), pp. 175–181. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(00)00142-3. URL: https://www.sciencedirect.com/science/article/pii/S0020019000001423.

Hermann, Karl Moritz et al. (2015). "Teaching Machines to Read and Comprehend." In: *CoRR* abs/1506.03340. arXiv: 1506.03340. URL: http://arxiv.org/abs/1506.03340.

Ivanova, Sardana and Valter Uotila (2021). *Finnish poetry generation using fine-tuned mBART*. URL: https://youtu.be/hux79WvZxYU (visited on 05/11/2022).

Jhamtani, Harsh et al. (Nov. 2019). "Learning Rhyming Constraints using Structured Adversaries." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics: Hong Kong, China, pp. 6025–6031. DOI: 10.18653/v1/D19-1621. URL: https://www.aclweb.org/anthology/D19-1621.

Julien, Marit (2021). *Norvagisering i Store norske leksikon på snl.no*. URL: https://snl.no/norvagisering (visited on 03/28/2022).

Lau, Jey Han et al. (July 2018). "Deep-speare: A joint neural model of poetic language, meter and rhyme." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics: Melbourne, Australia, pp. 1948–1958. DOI: 10.18653/v1/P18-1181. URL: https://www.aclweb.org/anthology/P18-1181.

Lear, Edward (2008). "Book of Nonsense." In: URL: https://www.gutenberg.org/files/982/982-h/982-h.htm (visited on 03/28/2022).

Lehmann-Haupt, H. E. (2020). "Johannes Gutenberg." In: URL: https://www.britannica.com/biography/Johannes-Gutenberg (visited on 02/21/2022).

Liao, Yi et al. (2019). "Gpt-based generation for classical chinese poetry." In: *arXiv preprint arXiv:1907.00151*.

Liu, Yusen, Dayiheng Liu, and Jiancheng Lv (2020). "Deep poetry: A chinese classical poetry generation system." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 09, pp. 13626–13627.

Malmi, Eric et al. (2015). "DopeLearning: A Computational Approach to Rap Lyrics Generation." In: *CoRR* abs/1505.04771. arXiv: 1505.04771. URL: http://arxiv.org/abs/1505.04771.

McHugh, Mary L (2012). "Interrater reliability: the kappa statistic." In: *Biochemia medica* 22(3), pp. 276–282.

Miller, George A (1995). "WordNet: a lexical database for English." In: *Communications of the ACM* 38(11), pp. 39–41.

Minogue, Paul (2021). *Using Siamese neural networks to create a simple rhyme detection system.* URL: https://paulminogue.com/index.php/2021/02/14/using-a-siamese-neural-network-to-create-a-simple-rhyme-detector/ (visited on 02/28/2022).

Nikolov, Nikola I. et al. (Dec. 2020). "Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders." In: *Proceedings of the 13th International Conference on Natural Language Generation.* Association for Computational Linguistics: Dublin, Ireland, pp. 360–373. URL: https://www.aclweb.org/anthology/2020.inlg-1.42.

Njåstad, Magne, Erik Opsahl, and Ida Scott (2022). *Dansketiden i Store norske leksikon på snl.no.* URL: https://snl.no/dansketiden (visited on 01/19/2022).

Nordbø, Børge and Hallvard Magerøy (2019). *Gammelnorsk i Store norske leksikon på snl.no.* URL: https://snl.no/gammelnorsk (visited on 03/24/2022).

Papineni, Kishore et al. (July 2002). "Bleu: a Method for Automatic Evaluation of Machine Translation." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics.* Association for Computational Linguistics: Philadelphia, Pennsylvania, USA, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040.

Radford, Alec, Karthik Narasimhan, et al. (2018). "Improving language understanding by generative pre-training." In:

Radford, Alec, Jeffrey Wu, et al. (2019). "Language models are unsupervised multitask learners." In: *OpenAI blog* 1(8), p. 9.

Raffel, Colin et al. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." In: *ArXiv* abs/1910.10683.

Ram, Naveen et al. (2021). "Say What? Collaborative Pop Lyric Generation Using Multitask Transfer Learning." In: *Proceedings of the 9th International Conference on Human-Agent Interaction*, pp. 165–173.

Saygin, Ayse Pinar, Ilyas Cicekli, and Varol Akman (2000). "Turing test: 50 years later." In: *Minds and machines* 10(4), pp. 463–518.

Stolpe Foss, Emmie (2022). *1 av 9 har nynorsk som hovudmål i skolen.* URL: https://www.ssb.no/utdanning/grunnskoler/statistikk/elevar-i-grunnskolen/artikler/1-av-10-har-nynorsk-som-hovudmal-i-skolen (visited on 02/28/2022).

Thorsnæs, Geir (2021). *Østlandet i Store norske leksikon på snl.no.* URL: https://snl.no/%5C%C3%5C%98stlandet (visited on 03/28/2022).

Tian, Yufei and Nanyun Peng (2022). "Zero-shot Sonnet Generation with Discourse-level Planning and Aesthetics Features." In: *arXiv preprint arXiv:2205.01821.*

Tikhonov, Aleksey and Ivan P. Yamshchikov (Oct. 2018). "Sounds Wilde. Phonetically Extended Embeddings for Author-Stylized Poetry Generation." In: *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology.* Association for Computational Linguistics: Brussels, Belgium, pp. 117–124. DOI: 10.18653/v1/W18-5813. URL: https://www.aclweb.org/anthology/W18-5813.

Van de Cruys, Tim (July 2020). "Automatic Poetry Generation from Prosaic Text." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* Association for Computational Linguistics: Online, pp. 2471–2480. DOI: 10.18653/v1/2020.acl-main.223. URL: https://www.aclweb.org/anthology/2020.acl-main.223.

Vaswani, Ashish et al. (2017). “Attention is all you need.” In: *arXiv preprint arXiv:1706.03762*.

Venås, Kjell, Dag Gundersen, and Børge Nordbø (2022). *Mellomnorsk i Store norske leksikon på snl.no*. URL: https://snl.no/mellomnorsk (visited on 01/19/2022).

Venås, Kjell and Børge Nordbø (2022). *Moderne norsk i Store norske leksikon på snl.no*. URL: https://snl.no/moderne_norsk (visited on 01/19/2022).

Venås, Kjell and Martin Skjekkeland (2022). *Dialekter i Norge i Store norske leksikon på snl.no*. URL: https://snl.no/dialekter_i_Norge (visited on 02/28/2022).

Vikør, Lars S (2022). *Rettskrivingsreforma av 1938 i Store norske leksikon på snl.no*. URL: https://snl.no/Rettskrivingsreforma_av_1938#-Gjennomf%5C%C3%5C%B8ring_og_resultat (visited on 03/28/2022).

Vikør, Lars S., Ernst Håkon Jahr, and Mikkel Berg-Nordlie (2022). *Språk i Norge i Store norske leksikon på snl.no*. URL: https://snl.no/spr%5C%C3%5C%A5k_i_Norge (visited on 02/28/2022).

Weidling, Tor Ragnar and Magne Njåstad (2022). *Norge under dansk styre - 1537-1814 i Store norske leksikon på snl.no*. URL: https://snl.no/Norge_under_dansk_styre_-_1537-1814 (visited on 01/19/2022).

Yang, Cheng et al. (2018). “Stylistic chinese poetry generation via unsupervised style disentanglement.” In: *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3960–3969.

Yi, Xiaoyuan et al. (2018). “Automatic poetry generation with mutual reinforcement learning.” In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3143–3153.

Zhang, Xingxing and Mirella Lapata (Oct. 2014). “Chinese Poetry Generation with Recurrent Neural Networks.” In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics: Doha, Qatar, pp. 670–680. DOI: 10.3115/v1/D14-1074. URL: https://www.aclweb.org/anthology/D14-1074.

Zugarini, Andrea, Stefano Melacci, and Marco Maggini (2019). “Neural poetry: Learning to generate poems using syllables.” In: *International Conference on Artificial Neural Networks*. Springer, pp. 313–325.

Zugarini, Andrea, Luca Pasqualini, et al. (2021). “Generate and Revise: Reinforcement Learning in Neural Poetry.” In: *CoRR* abs/2102.04114. arXiv: 2102.04114. URL: https://arxiv.org/abs/2102.04114.

Appendices

Appendix A

Repairing the "bad clusters" from rhyme pair graph

Below are descriptions of the automatic and manual clustering of the 27 "bad" connected components from the rhyme pair graph described in chapter 4.

A.1 Largest connected component

In Figure A.1 we see the largest connected component, consisting of 227 vertices.

In Figure A.3 we see the result of manually separating the graph in Figure A.1. 19 edges were removed from the original graph and 2 edges were added, resulting in 11 connected components of size >1. The components contain words with the phonetic endings (from left to right, top to bottom) ɔrd, æjɛ, uːt, æɾɛ, uːɾ, uʈ, ɑnː, uː, ʉɛɾ, ɔʈ and æɾɛt. 1 word ("takke") was separated into a single-vertex component.

9 of the removed edges were derived from obviously wrong rhyme scheme annotations, the word pairs having little to no phonetic similarity. 2 of the edges removed contained the word "*været*" (væɾɛt) 'been', ("vært" (væʈ) in modern Norwegian). These were removed to seperate them from the more usual heteronym "*været*" (væɾɛ) 'the weather'. The remaining 6 edges that were removed were edges between words with the phonetically similar word-endings uːt - uːd and uʈ - oʈ.

In Figure A.2 we see the result of running this through the HCS clustering algorithm. The graph was separated into 8 connected components of size >1, consisting of 50 vertices in total. 177 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

The clusters contain words with the phonetic endings uːɾ (top center), æjɛ (top right), uː (mid left), ɑnː (mid center), æɾɛ (mid right), uʈ (bottom left), ʉɛɾ (bottom center) and uːt (bottom right). The ɔrd-, ɔʈ- and æɾɛt-clusters from the manual clustering were not extracted by the HCS algorithm.

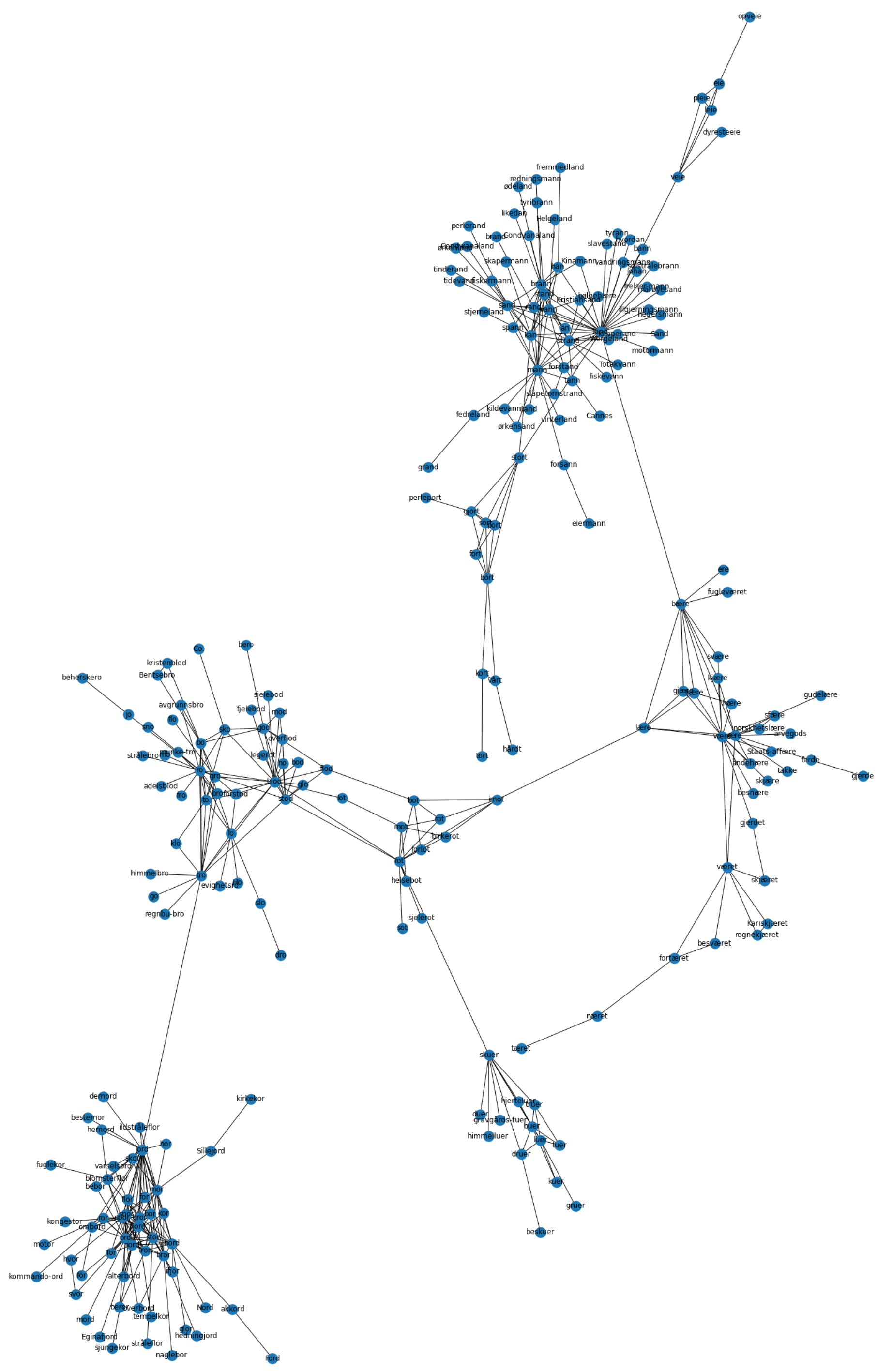


Figure A.1: The largest connected component

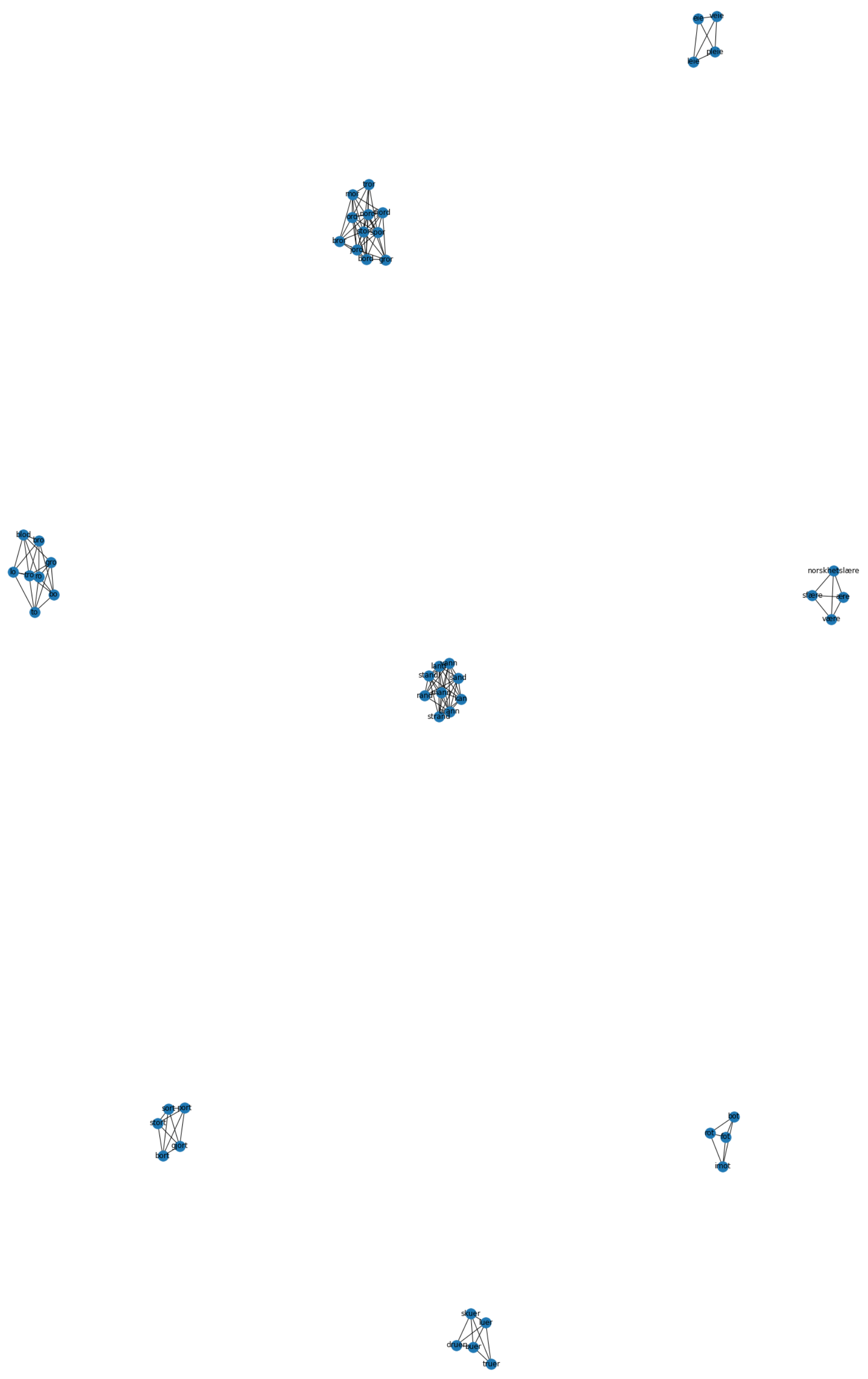


Figure A.2: Post-clustering largest connected component (single vertices omitted)

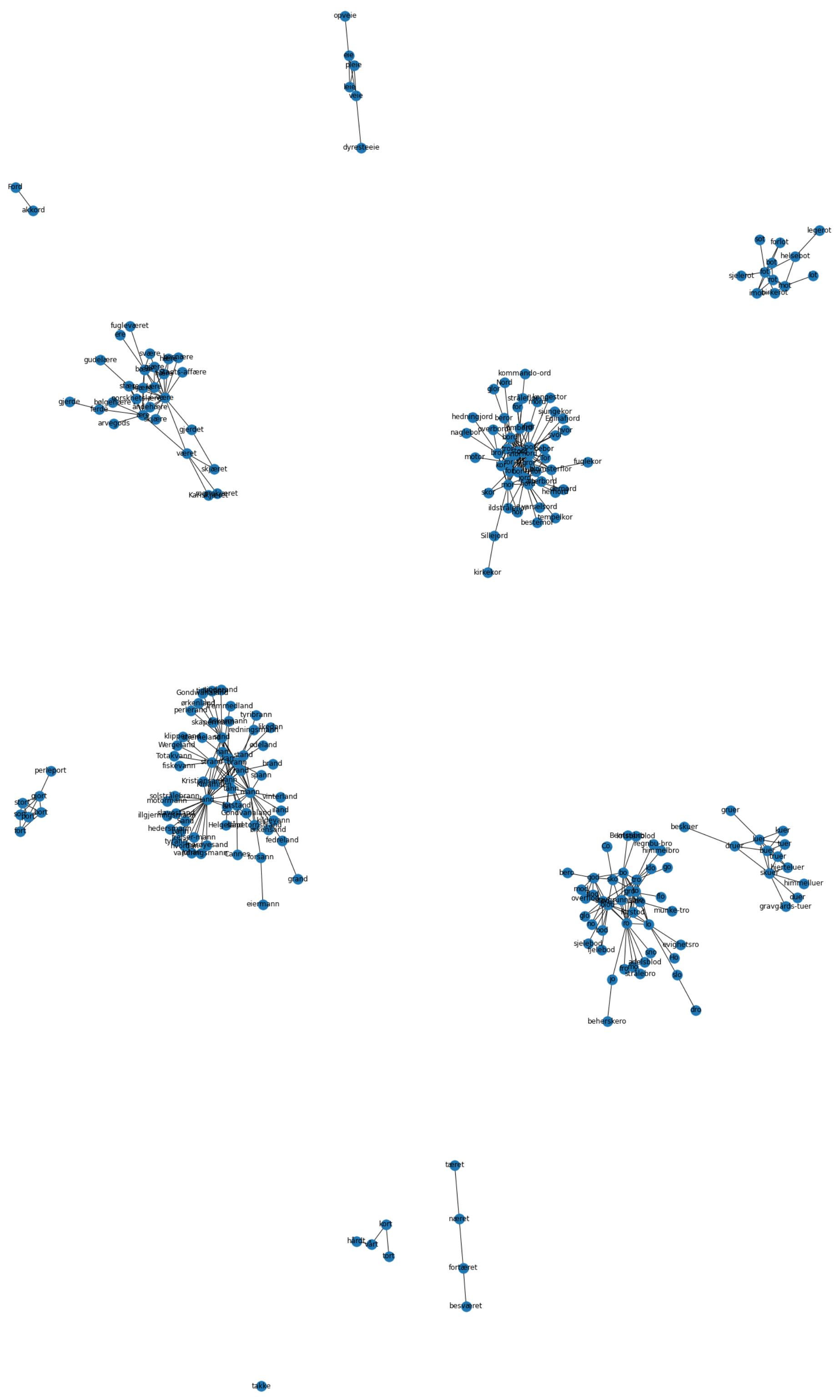


Figure A.3: Manual clustering largest connected component

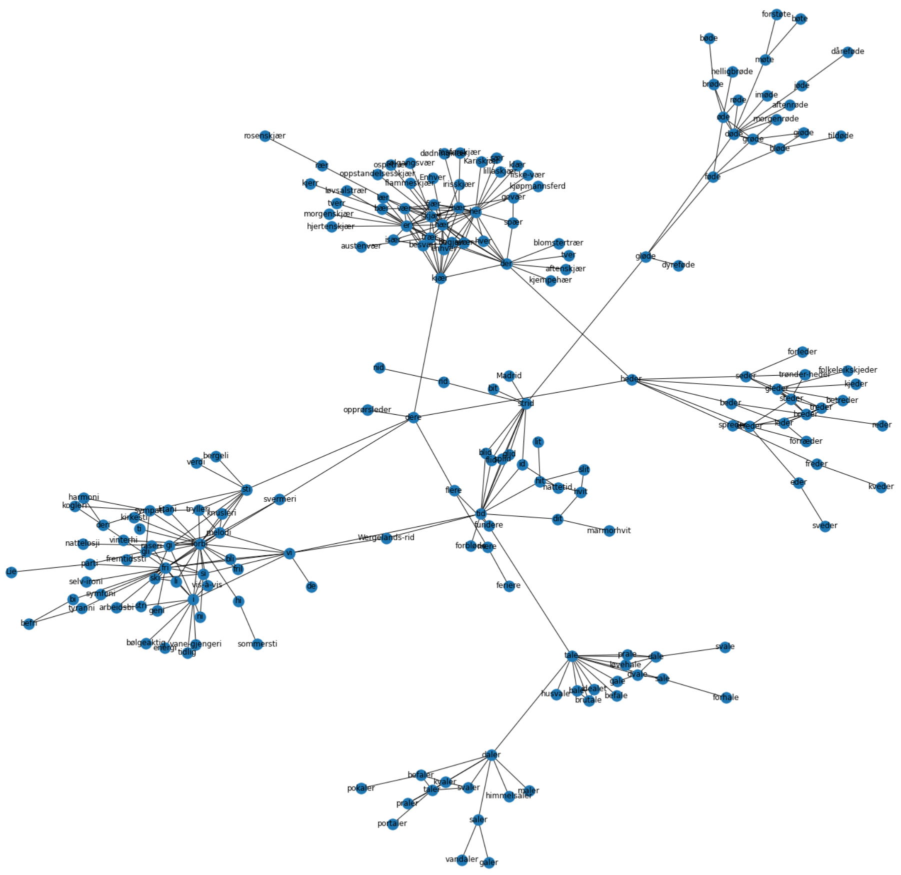


Figure A.4: 2nd largest connected component

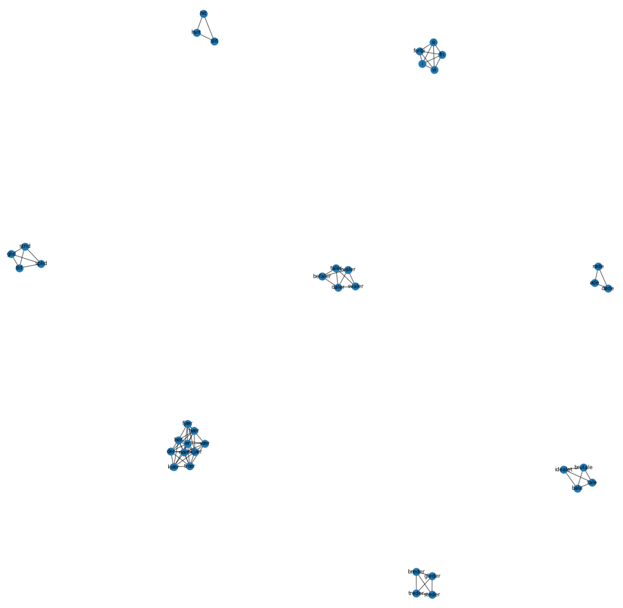


Figure A.5: Post-clustering 2nd largest connected component (single vertices omitted)

A.2 Second largest connected component

In Figure A.4 we see the second largest connected component, consisting of 189 vertices. In Figure A.5 we see the result of running this through the HCS clustering algorithm. The graph was separated into 8 connected components of size >1, consisting of 38 vertices in total. 151 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs. The components contain words ending with iːt (top left), iː (top right), iːd (mid left), ɑlɛɾ (mid center), øːdɛ (mid right), æːɾ (bottom left), ɛdɛɾ (bottom center) and ɑlɛ (bottom right).

In Figure A.6 we see the result of manually separating the graph in Figure A.4. 16 edges were removed and 2 edges were added, resulting in 10 connected components of size >1. The components contain words with the phonetic endings ɑlɛ (top left), æːɾ (top center), øːtɛ (top left), øːdɛ (mid left), ɑlɛɾ (mid left-center), ɛdɛɾ (mid right-center), iː (mid right), iːd (bottom left), iːt (bottom center) and ɛɾɛ (bottom right).

1 word ("forbløde") was separated into a single-vertex component. The øːtɛ- and ɛɾɛ-id clusters were only extracted by the manual clustering, ie they were not extracted with the HCS algortihm.

Of the 15 removed edges, 9 of were derived from obviously wrong rhyme scheme annotations, with little similarity in the phonology of the words in the word pair. 5 of the edges derived from phonologically similar word pairs, with word pairs of words ending with (øde - øte), (id - it) and (i - id).

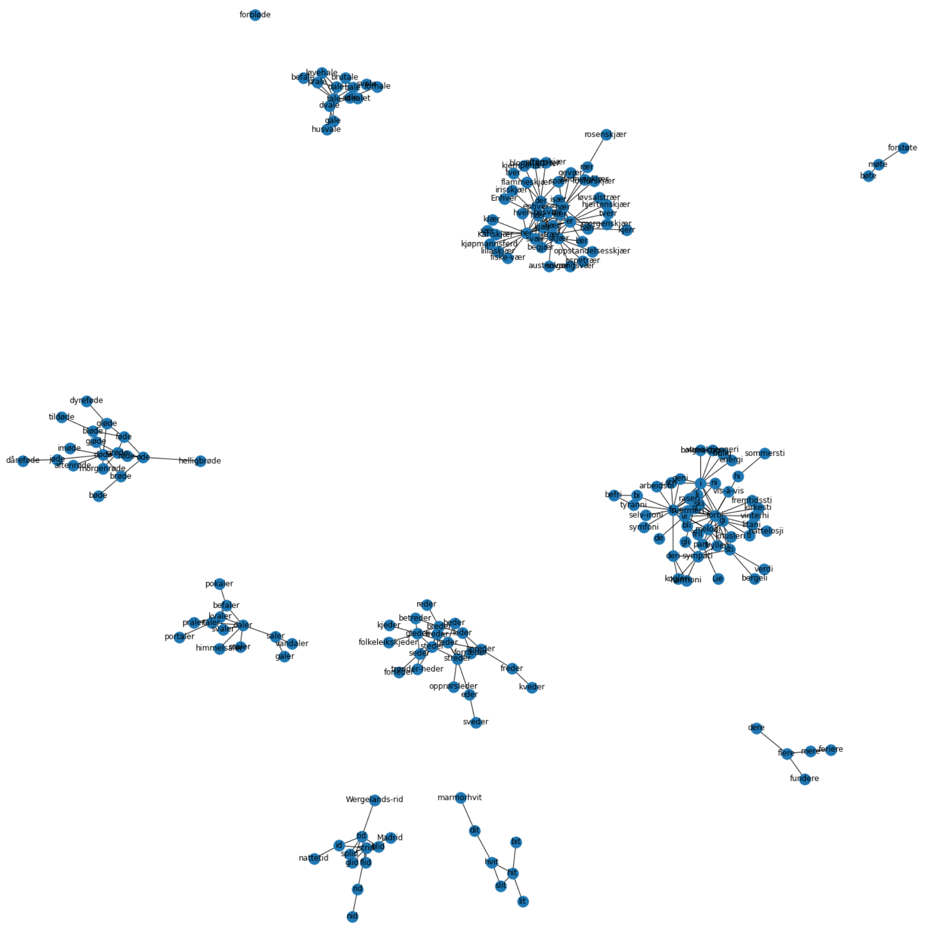


Figure A.6: Manual clustering 2nd largest connected component

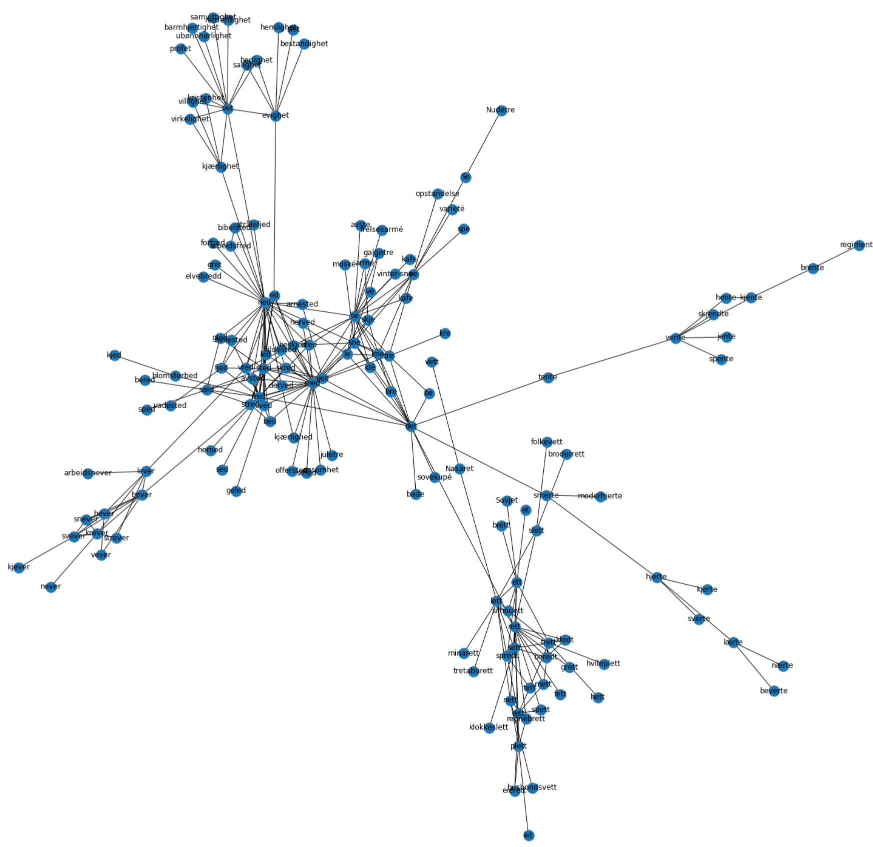


Figure A.7: 3rd largest connected component

The components contain words ending with and .

A.3 Third largest connected component

In Figure A.7 we see the third largest connected component from the graph representation of the annotated rhyme pairs, consisting of 148 vertices. In Figure A.8 we see the result of running this through the HCS clustering algorithm. The graph was separated into 5 connected components of size >1, consisting of 23 vertices in total. The components contain words ending with ɛntə (top left), ɛvɛr (top right), ɛːd (left), ɛːt (right), and ɛtː (bottom). 125 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.9 we see the result of manually separating the graph in Figure A.7. 11 edges were removed from the original graph, resulting in 6 connected components of size >1. 3 vertices were separated into single-vertex components,

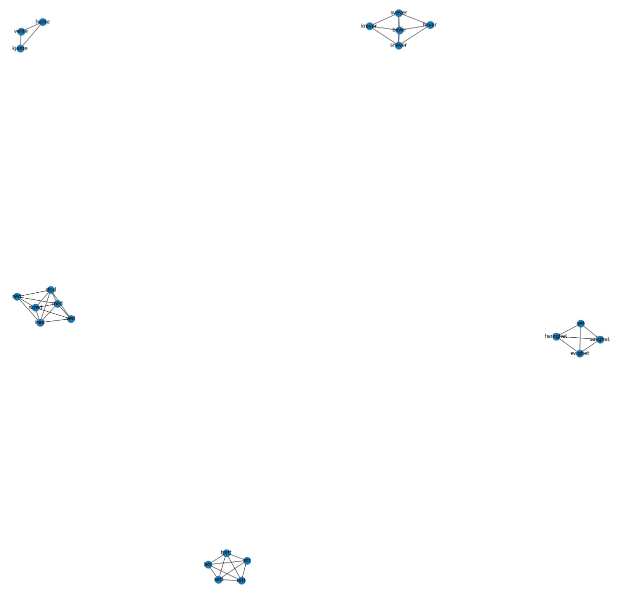


Figure A.8: Post-clustering 3rd largest connected component

and could thus not be used to create more rhyme pairs. The components contain words ending with ɛːd/ɛː (top left), ɛtː (top right), ɛvɛɾ (mid left), ɛntə (bottom left), æʈə (bottom center), and ɛːt (bottom right).

3 of the removed edges were not phonetically similar and likely a result of mistakenly annotating the wrong rhyme scheme. 3 of the removed edges were from multiword rhymes. The edges were ("tente", "det"), ("smerte", "det") and ("bade", "det"), and the target rhymes are ("tente", "sendt det"), ("smerte", "lært det") and ("bade", "fra det"). The remaining edges that were removed were between words with the similar phonetic endings ɛːd - ɛːt and ɛːd - ɛdː.

A.4 Fourth connected component

In Figure A.10 we see the 4th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 144 vertices.

In Figure A.11 we see the result of running this through the HCS clustering algorithm. The graph was separated into 5 connected components of size >1, consisting of 38 vertices in total. The components contain words ending with ʉː (top left), oː (bottom left), oːr (center), uɾdn̩/uːun̩ (top right) and ɛst (bottom right). 106 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.12 we see the result of manually clustering the graph in Figure A.10. 11 edges were removed from the original graph, resulting in 8 connected components. The components contain words ending with oːt (top left), oː (mid

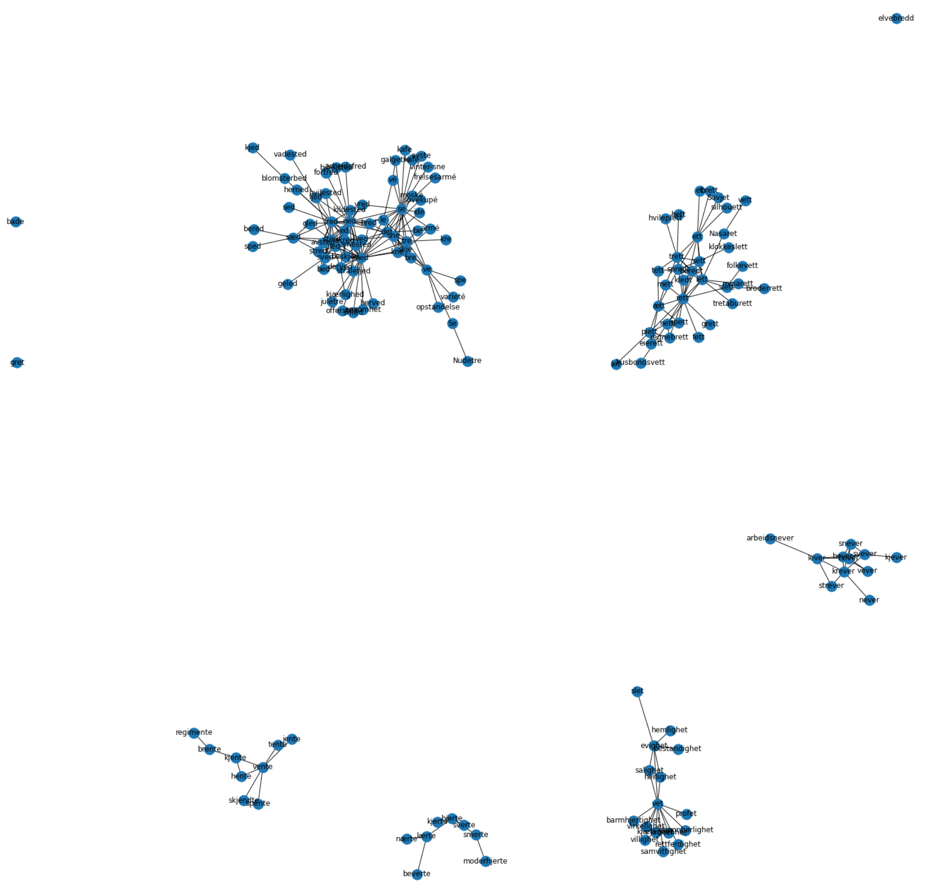


Figure A.9: Manual clustering 3rd largest connected component

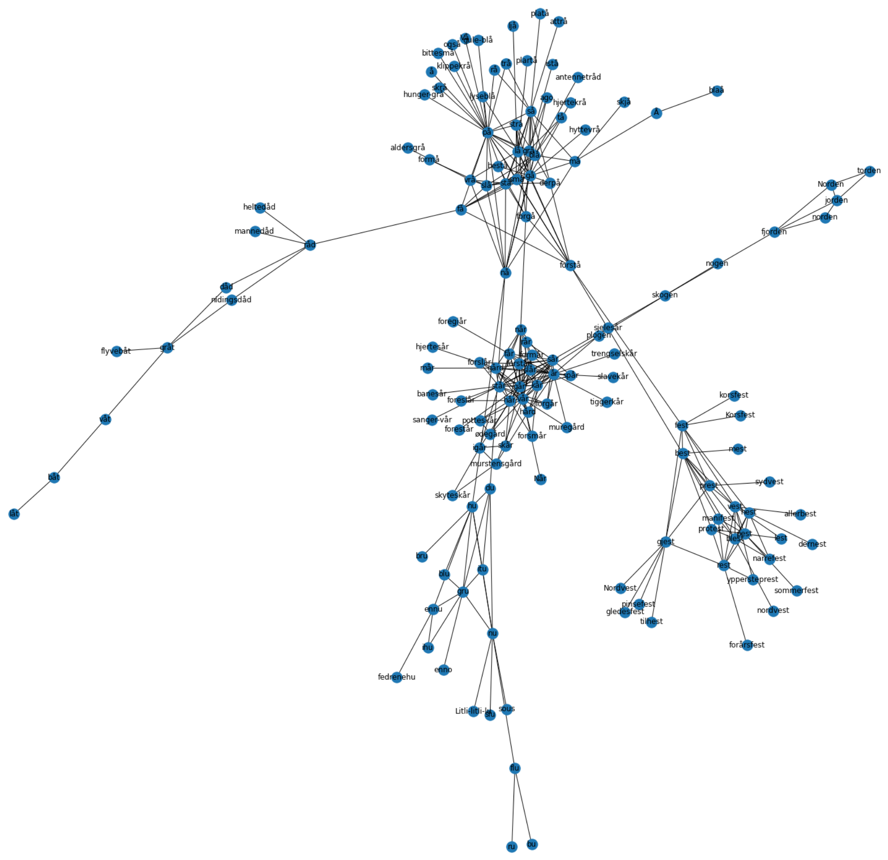


Figure A.10: 4th largest connected component

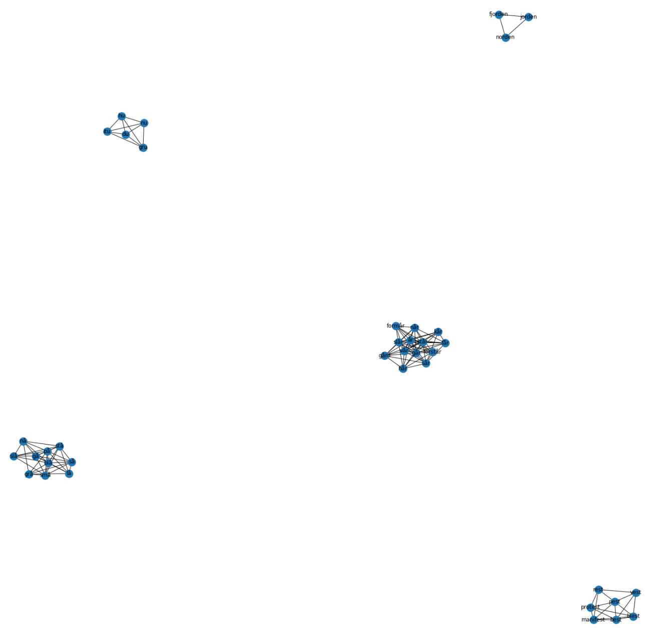


Figure A.11: Post-clustering 4th largest connected component

left), ʉː (bottom left), ɛst (top center), oːr (mid center), oːd (bottom center), uɾdn̩/uːun̩ (top right), and uːgən (bottom right).

6 of the removed edges are between words that do not rhyme at all. 2 of the removed edges, ("du", "nå") and ("hu", "nå"), pair "nå" (noː) with words ending with ʉ. This is likely due to the word "nu" being replaced with "nå" during the Norwegifying step described in chapter 3. 2 of the edges that were removed were pairs where "gråt" (gɾoːt) was paired with a word that ends with oːd. Gråt is written gråd in Danish, so this is probably a trace of this. Finally, the edge ("råd", "få") was removed. "Råd" can be pronounced both as ɾoːd and roː, so this edge had to be removed to keep the oːd and oː clusters separate.

A.5 Fifth largest connected component

In Figure A.13 we see the 5th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 92 vertices. In Figure A.14 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 16 vertices in total. 76 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.15 we see the result of manually separating the graph in Figure A.13. 5 edges were removed from the original graph, resulting in 3 connected components of size >1. The components contain words ending with ɑːk, ɑːg

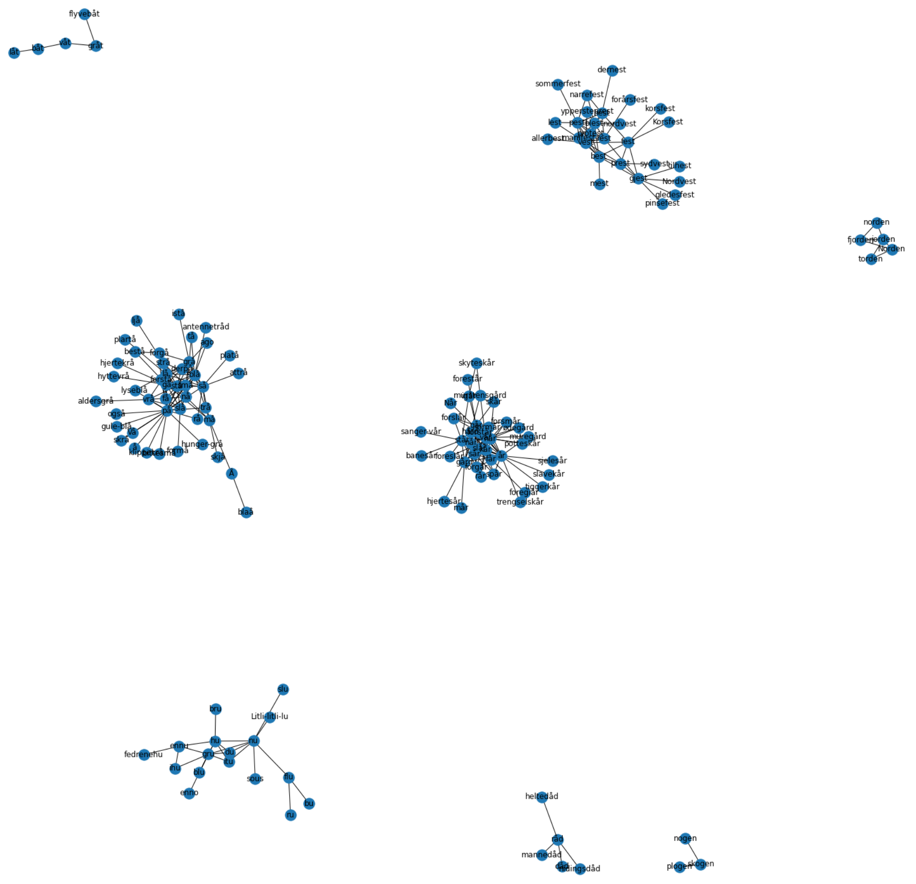


Figure A.12: Manual clustering 4th largest connected component

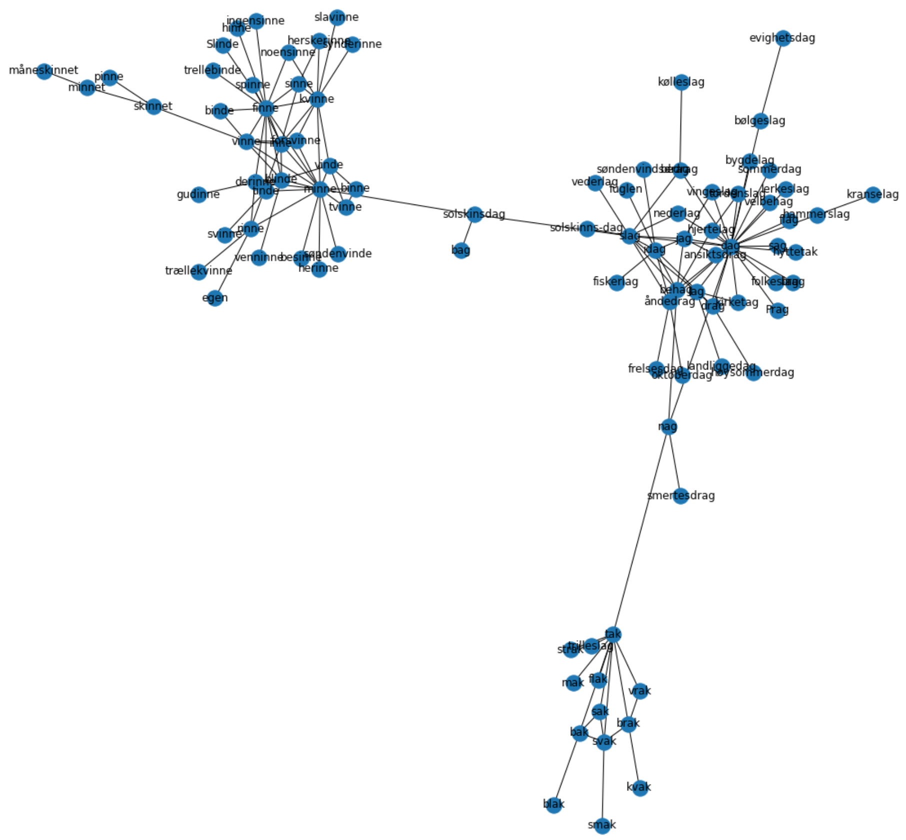


Figure A.13: 5th largest connected component

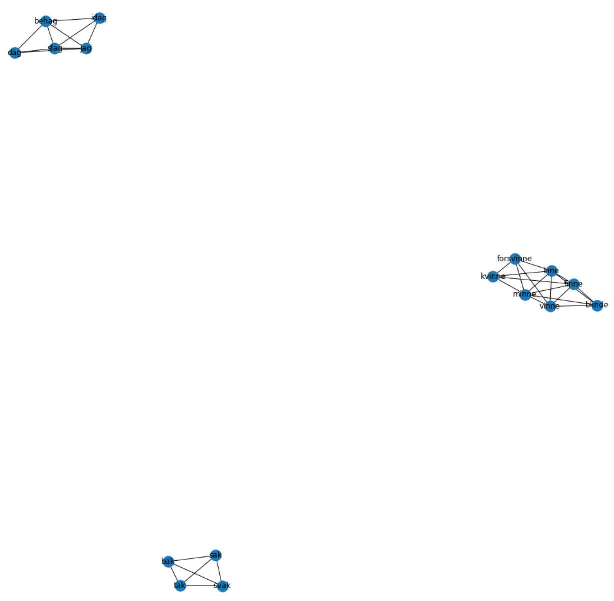


Figure A.14: Post-clustering 5th largest connected component

and ɪnːe. 3 of the removed edges were between words that are not phonetically similar. The other 2 were between words ending with ɑːk and ɑːg. 2 vertices were separated into single-vertex components, and could thus not be used to create more rhyme pairs.

A.6 Sixth largest connected component

In Figure A.16 we see the 6th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 68 vertices.

In Figure A.18 we see the result of manually separating the graph in Figure A.16. 7 edges were removed from the original graph, resulting in 4 connected components of size >1. One vertex was separated into a single-vertex component. The connected components contain words ending with ɛlv (top), omːər (left), ɛːl (center), and ɛlː (bottom).

3 of the 7 edges removed were simply wrong; ("forfengelighet", "selv"), ("del", "trommer"), ("sjel", "dommer"). 3 of the edges removed contained the word "selv (sɛlː paired with words ending with ɛlv. The last edge was ("sjel", "vell"), where "sjel" has a long vowel, and "vell" a short.

In Figure A.17 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 11 vertices in total. 57 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs. Among these are all words ending with ɛlv. The connected

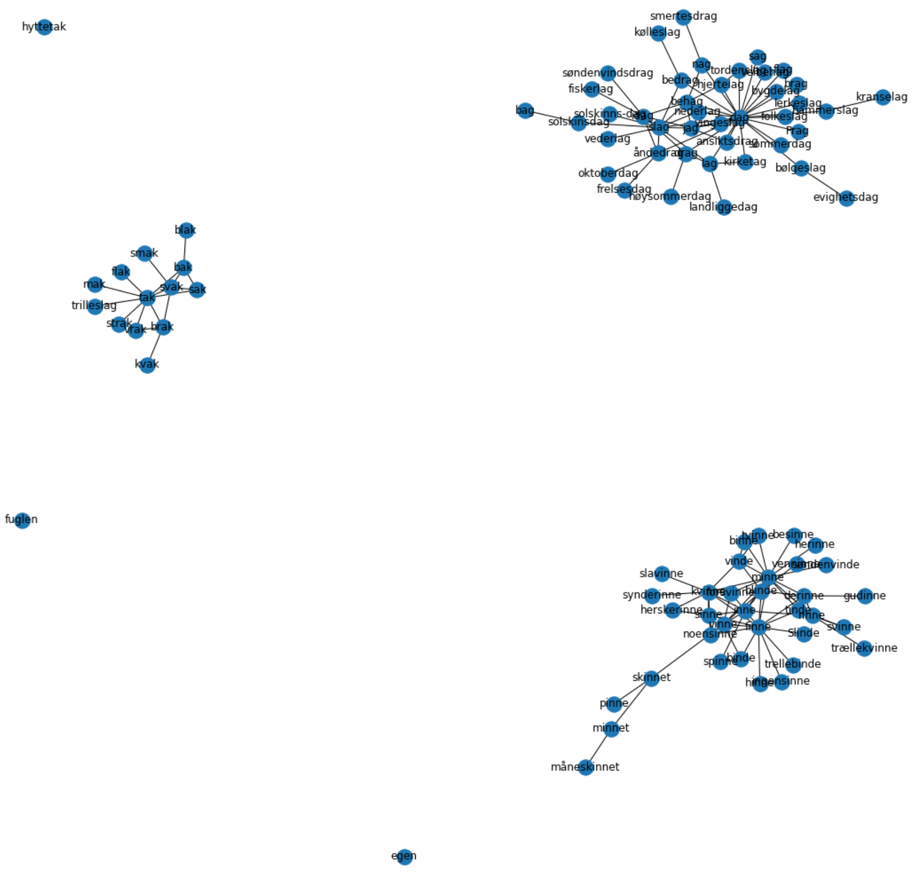


Figure A.15: Manual clustering 5th largest connected component

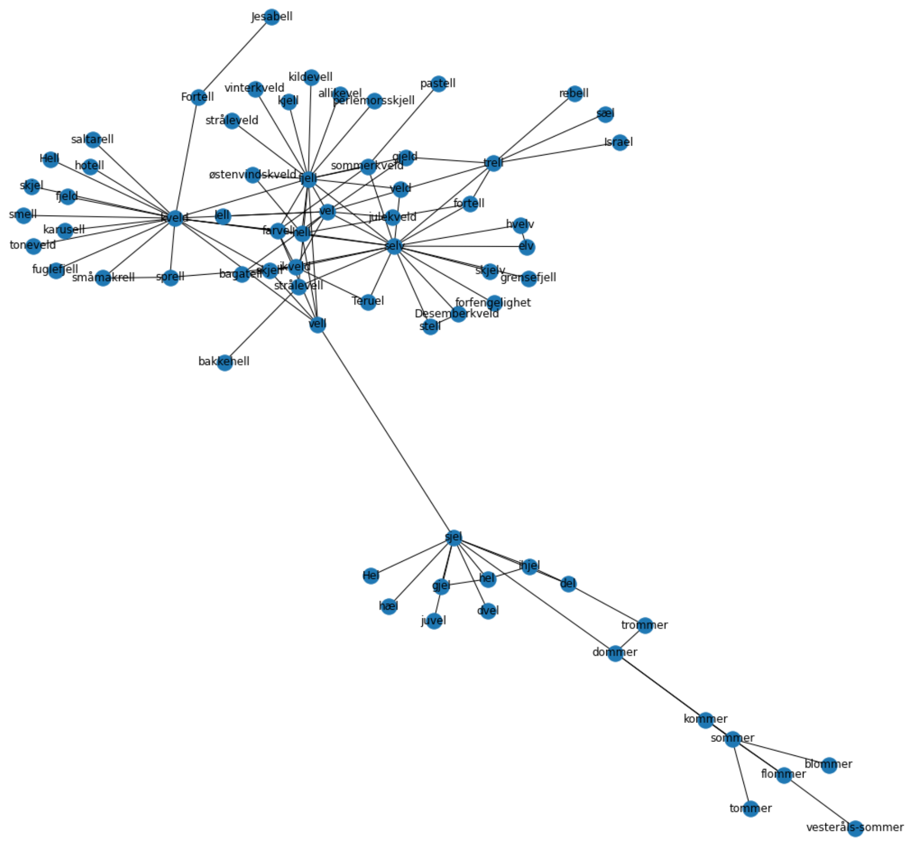


Figure A.16: 6th largest connected component

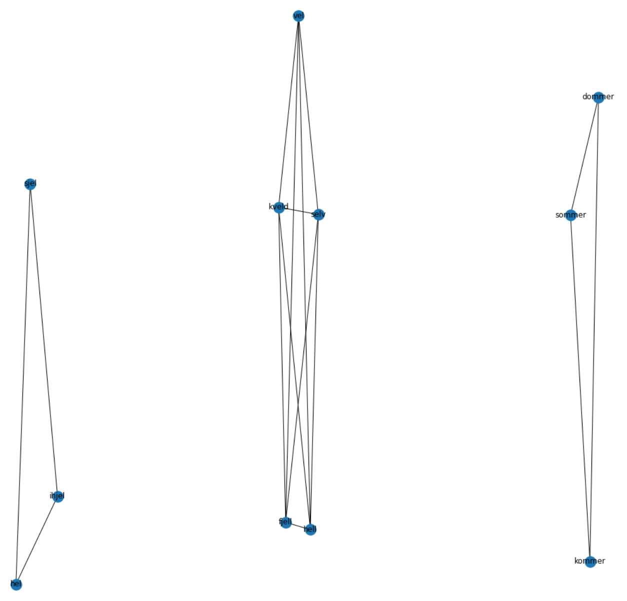


Figure A.17: Post-clustering 6th largest connected component

components we see in Figure A.17 contain words ending with ɛːl (left), ɛlː (center) and ɔmːər (right).

A.7 Seventh largest connected component

In Figure A.19 we see the 7th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 67 vertices. In Figure A.20 we see the result of running this through the HCS clustering algorithm. The graph was separated into 4 connected components of size >1, consisting of 19 vertices in total. 48 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.21 we see the result of manually separating the graph in Figure A.19. 11 edges were removed from the original graph, resulting in 5 connected components of size >1. 2 vertices were separated into single-vertex components, and could thus not be used to create more rhyme pairs. The components contain words ending with eːn (top), enː (centre), aŋkeɾ (centre right), iːdn̩ (right) and ɑtːn̩ (bottom).

4 of the removed edges (("ranker", "venn"), ("tanker", "ben"), ("anker", "én") and ("tanker", "menn")) were between words with no phonetic similarity, and are results of mistakenly writing the wrong rhyme scheme during annotation.

4 of the removed edges were from rhymes that span several words, all con-

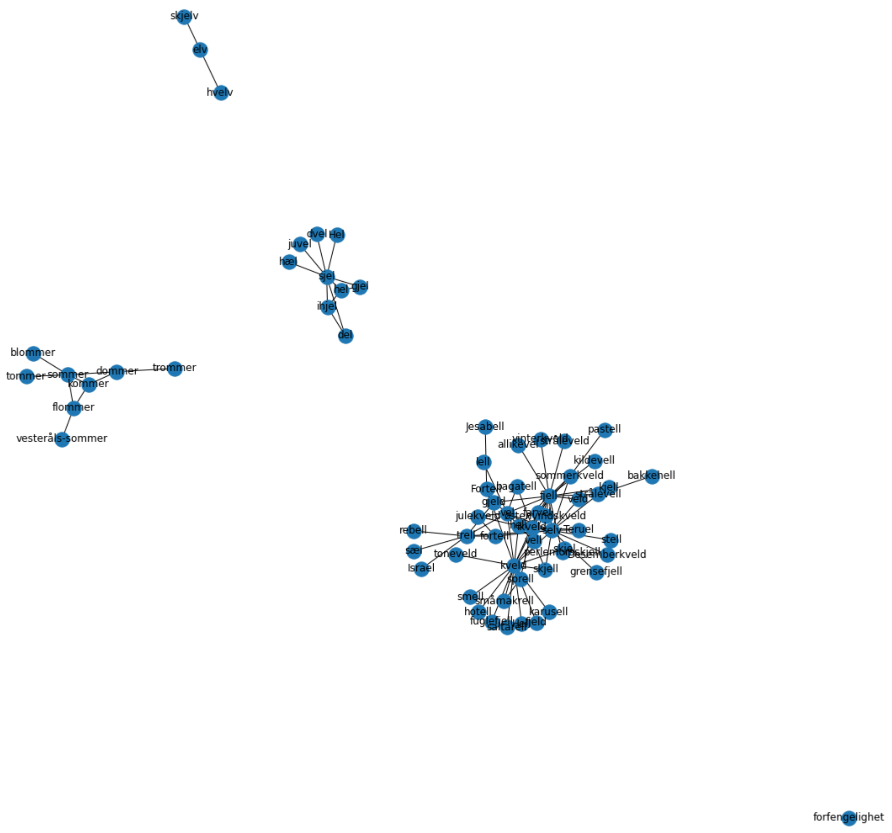


Figure A.18: Manual clustering 6th largest connected component

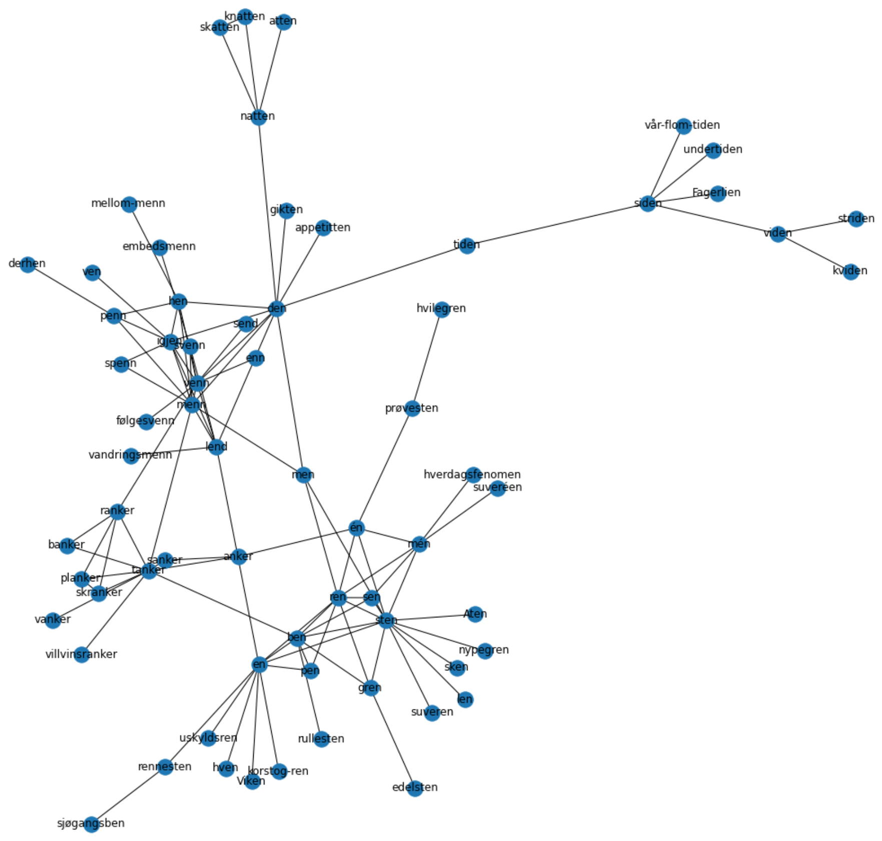


Figure A.19: 7th largest connected component

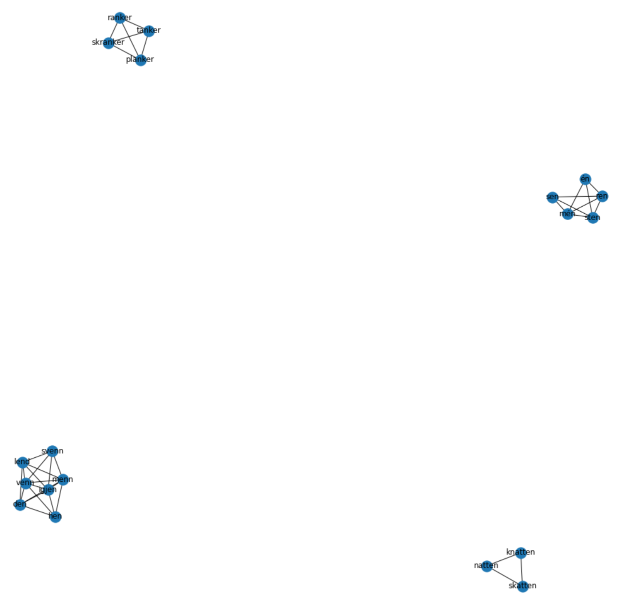


Figure A.20: Post-clustering 7th largest connected component

taining the word "den". These edges are ("natten", "den"), ("tiden", "den"), ("appetitten", "den") and ("gikten", "den"). The rhymes the pairs stem from are ("natten", "etterlatt den"), ("tiden", "i den"), ("appetitten", "kvitt den") and ("gikten", "likt den").

The last 3 edges that were removed were to separate the eːn and enː clusters. These contain hypernyms such as the letter sequence "men", which can be pronounced mɛnː 'but' or meːn 'injury'.

A.8 Eighth largest connected component

In Figure A.22 we see the 8th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 62 vertices. In Figure A.23 we see the result of running this through the HCS clustering algorithm. The graph was separated into 4 connected components of size >1, consisting of 16 vertices in total. The connected components contain words ending in ɑːd (left), ɑː (top and right) and ɑːv (bottom). 46 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.24 we see the result of manually separating the graph in Figure A.22. The graph is separated into 3 components, ending with ɑː (top), ɑːd (left) and ɑːv (right). This graph contains several words with multiple pronunciations. "Av" can be pronounced both ɑː and ɑːv. "Glad" is usually pronounced glɑː, but has in this data set been in several rhyme pairs with words ending with

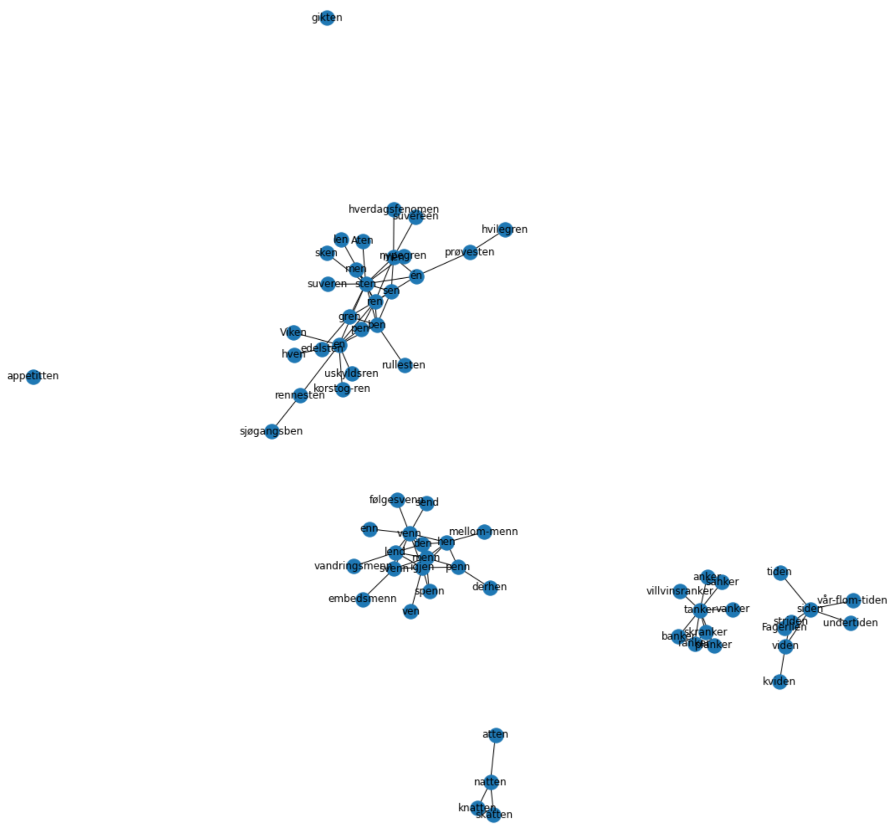


Figure A.21: Manual clustering 7th largest connected component

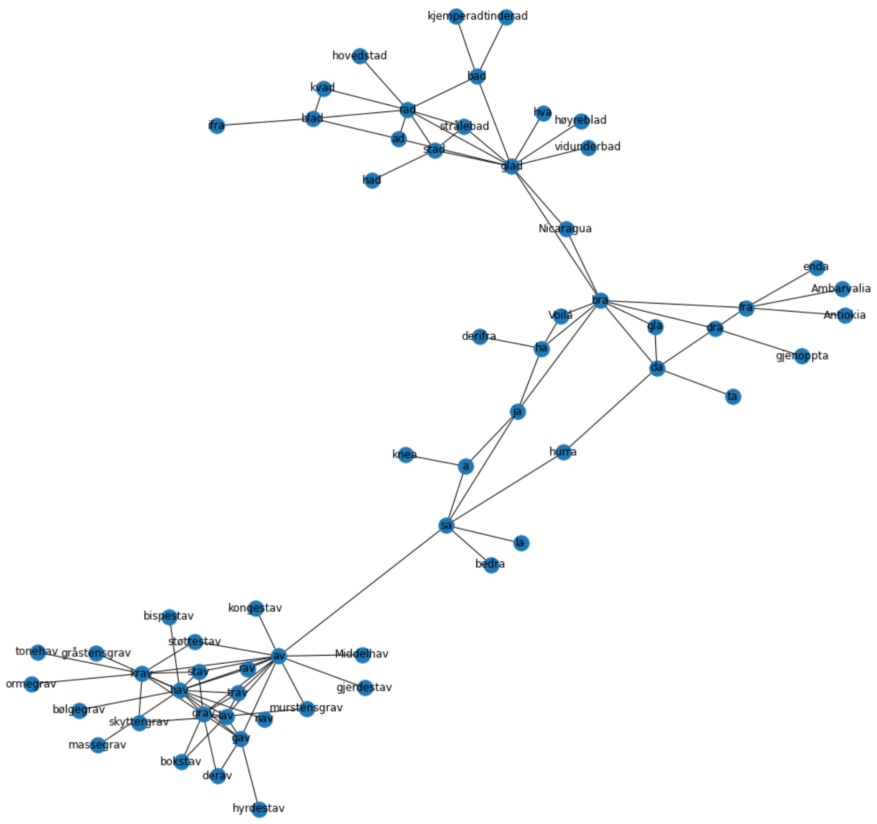


Figure A.22: 8th largest connected component

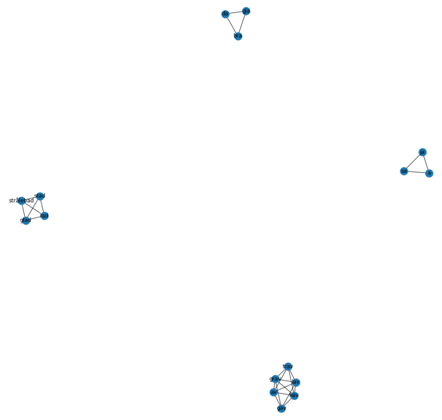


Figure A.23: Post-clustering 8th largest connected component

ɑːd. "Blad", and "stad" are pronounced both with ɑː and ɑːd word endings depending on the speaker and the context.

9 edges were removed from the original graph, and 3 edges were added. None of the removed edges were "wrong" per se, but contained the words mentioned above and thus connected graphs that contain words that do not rhyme. The added edges were to reconnect separated vertices to their respective components.

A.9 Ninth largest connected component

In Figure A.25 we see the 9th largest connected component from the graph representation of the annotated rhyme pairs, consisting of 58 vertices. In Figure A.26 we see the result of running this through the HCS clustering algorithm. The graph was separated into 9 connected components of size >1, consisting of 10 vertices in total. 48 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.27 we see the result of manually separating the graph in Figure A.25. 2 edges were removed from the original graph, resulting in 2 connected components. No vertices were separated into single-vertex components.

The components contain words ending with ɑkt and ʏtːɛ. The two removed edges were ("hytte", "makt") and ("beskytte", "prakt"), which were added as a result of a mistake during rhyme scheme annotation.

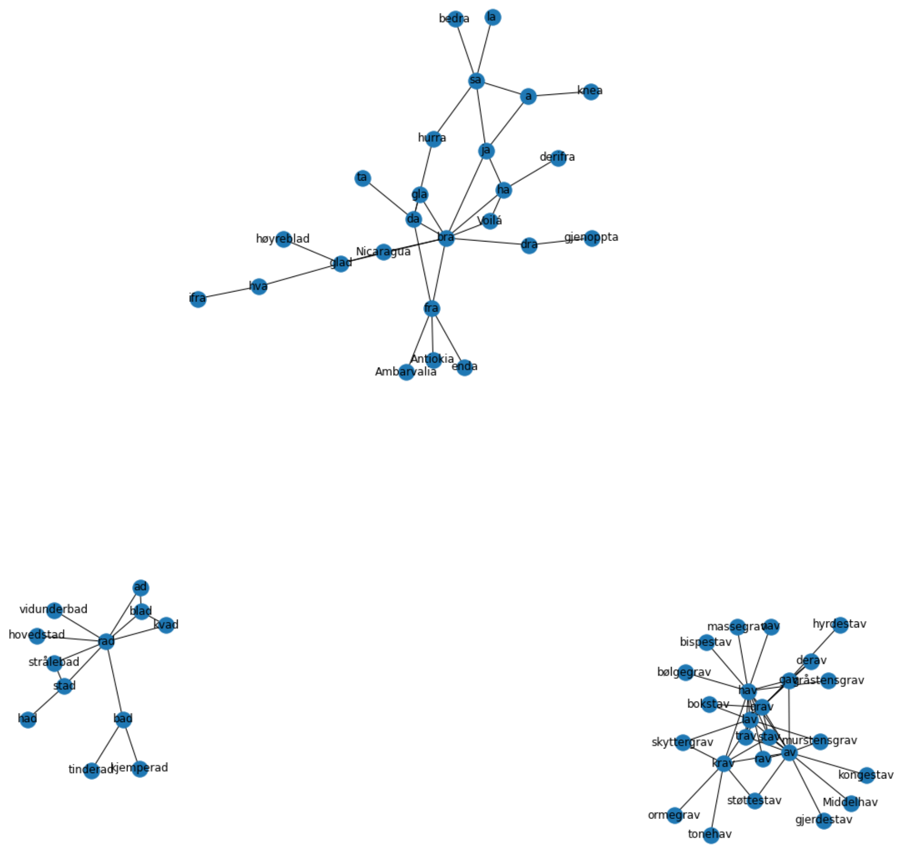


Figure A.24: Manual clustering 8th largest connected component

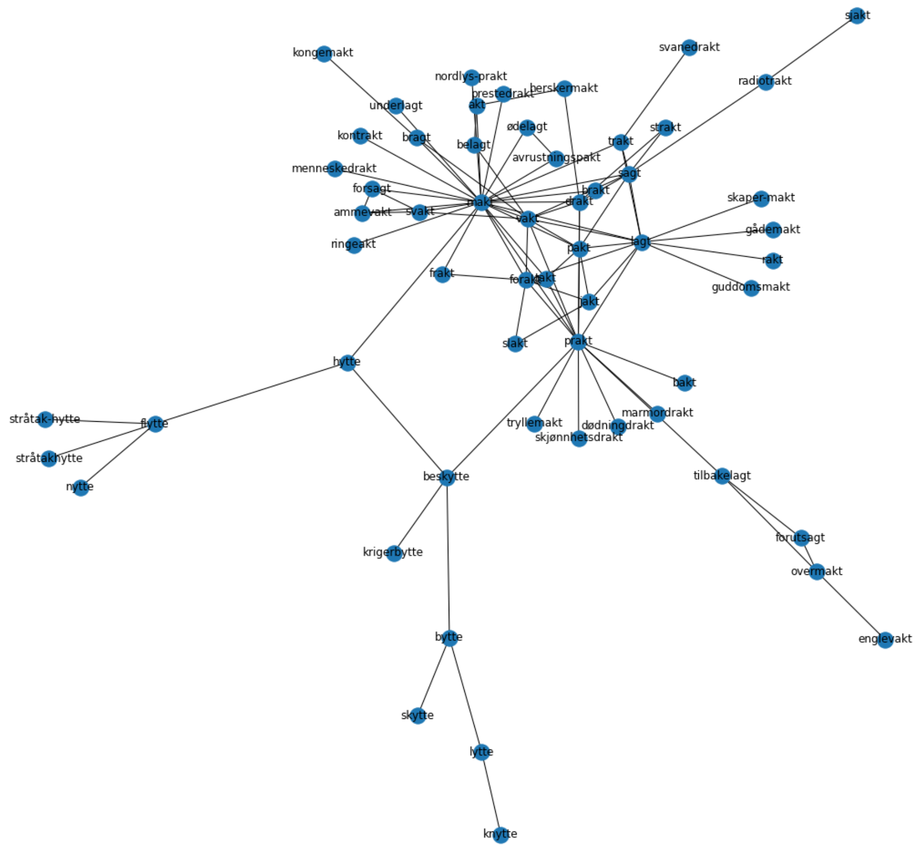


Figure A.25: 9th largest connected component

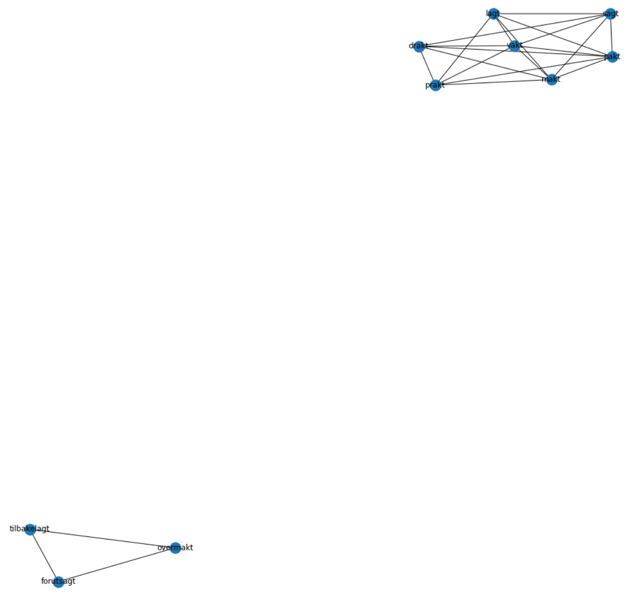


Figure A.26: Post-clustering 9th largest connected component

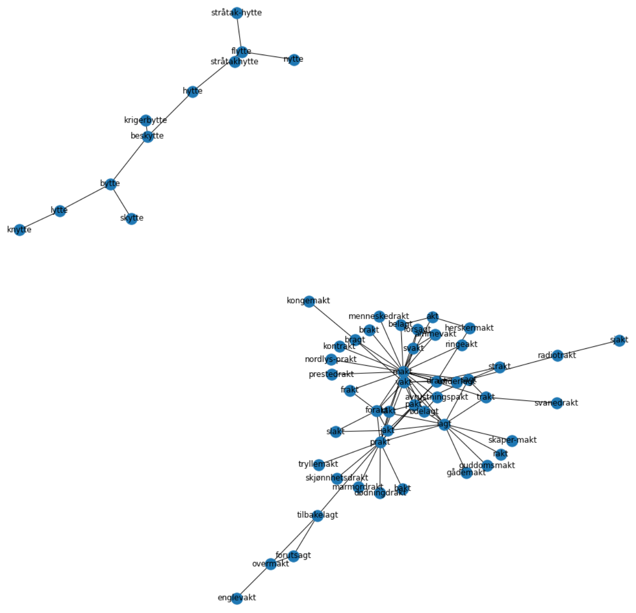


Figure A.27: Manual clustering 9th largest connected component

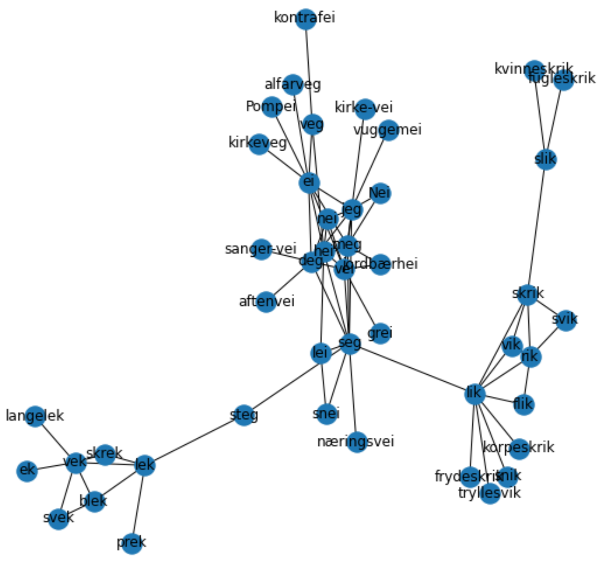


Figure A.28: Connected component A

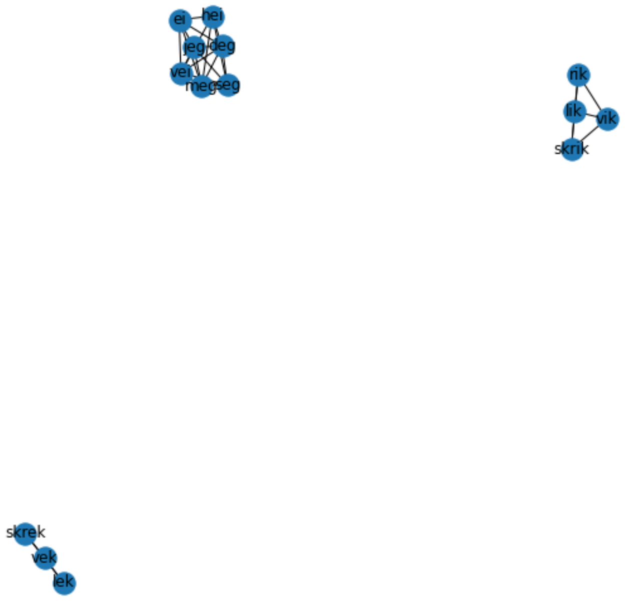


Figure A.29: Post-clustering connected component A

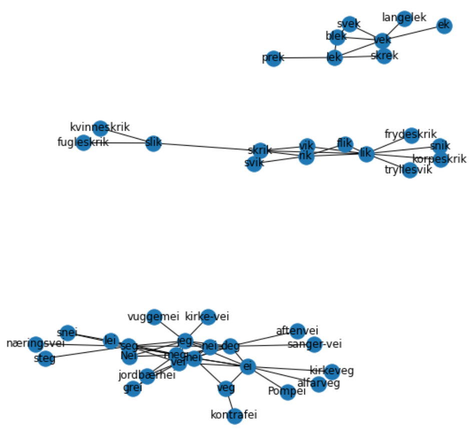


Figure A.30: Manual clustering connected component *A*

A.10 Connected component *A*

In Figure A.28 we see another bad connected component from the annotated rhyme pair graph, here named *A*. It consists of 45 vertices. In Figure A.29 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 14 vertices in total. 31 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.30 we see the result of manually separating the graph in Figure A.28. 2 edges were removed from the original graph, resulting in 3 connected components of size >1. The components contain words ending with ɛːk (left), æj and iːk. No vertices were separated into single-vertex components.

A.11 Connected component *B*

In Figure A.31 we see another bad connected component from the annotated rhyme pair graph, here named *B*. It consists of 45 vertices.

In Figure A.32 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 12 vertices in total. 33 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.33 we see the result of manually separating the graph in Figure

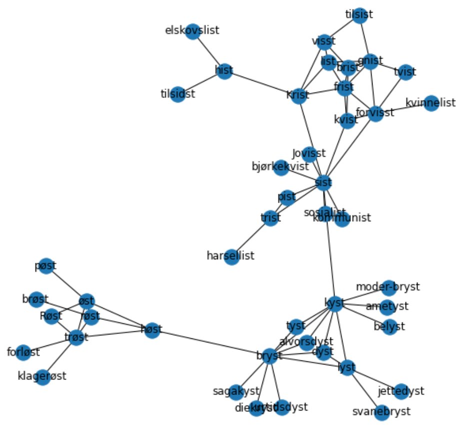


Figure A.31: Connected component *B*

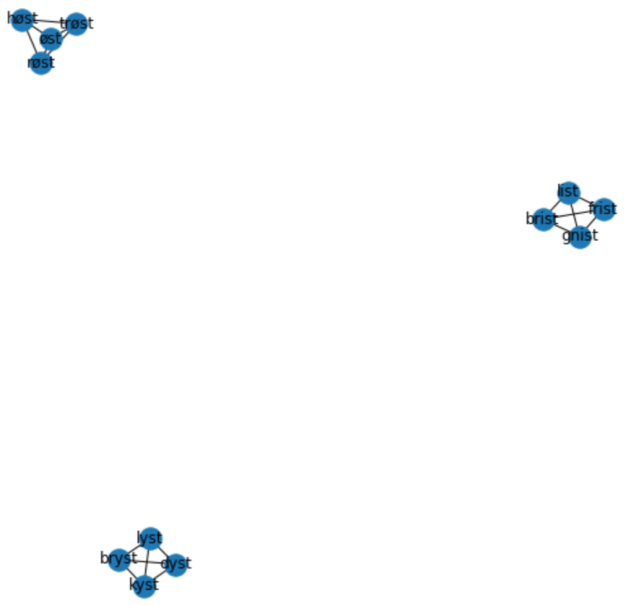


Figure A.32: Post-clustering connected component *B*

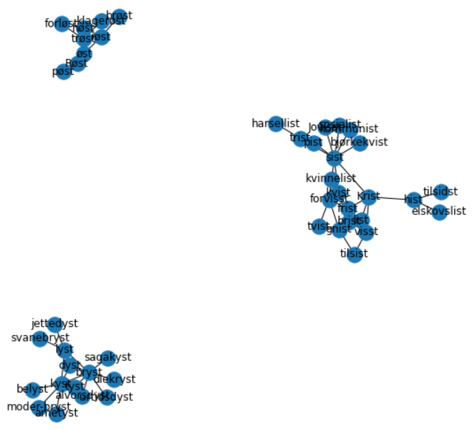


Figure A.33: Manual clustering connected component B

A.31. 2 edges were removed from the original graph, resulting in 3 connected components of size >1. The components contain words ending with øst, ʏst and ɪst. No vertices were separated into single-vertex components.

A.12 Connected component C

In Figure A.34 we see another connected component from the annotated rhyme pair graph, here named C. It consists of 39 vertices.

In Figure A.35 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 11 vertices in total. 29 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.36 we see the result of manually separating the graph in Figure A.34. 2 edges were removed from the original graph, resulting in 2 connected components. No vertices were separated into single-vertex components. The components contain words ending in ɛnːɛr (left) and ɑmː (right).

The removed edges are ("ham", "sender") and ("ham", "hender"), results of a wrongly annotated stanza.

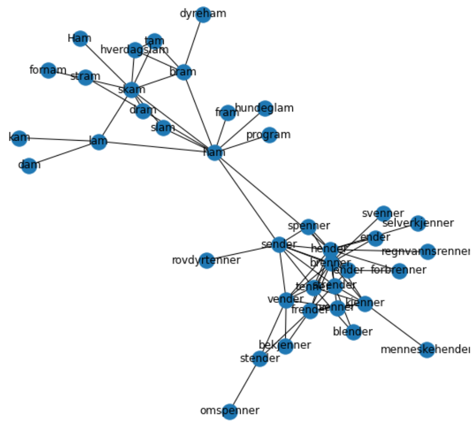


Figure A.34: Connected component C

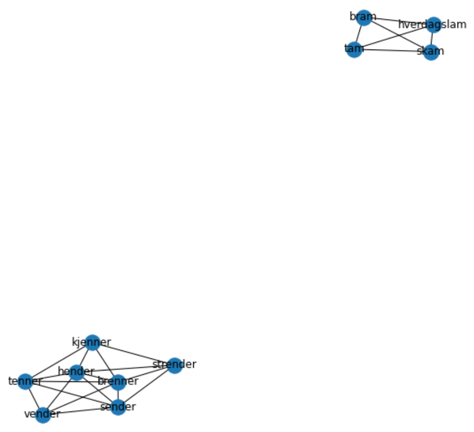


Figure A.35: Post-clustering connected component C

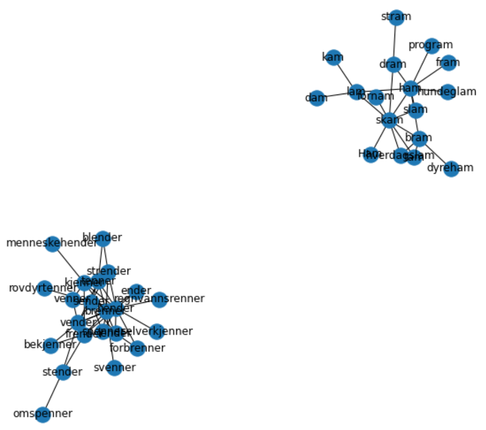


Figure A.36: Manual clustering connected component C

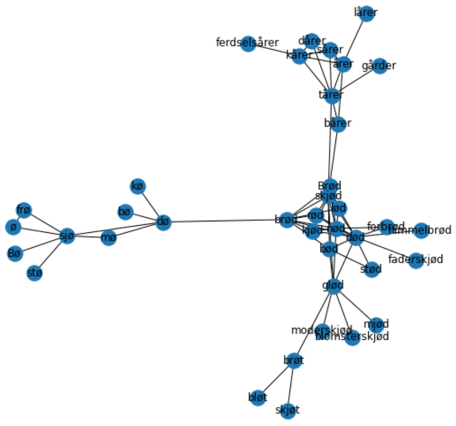


Figure A.37: Connected component D

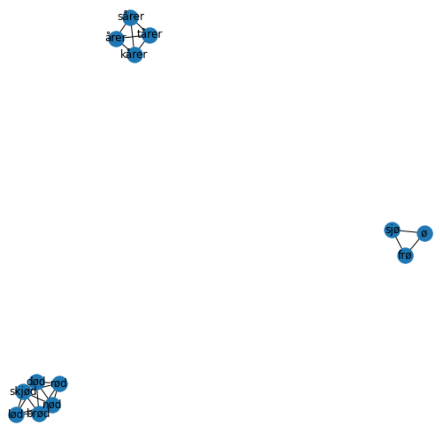


Figure A.38: Post-clustering connected component D

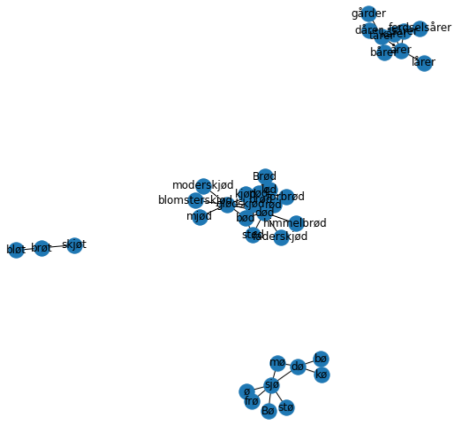


Figure A.39: Manual clustering connected component D

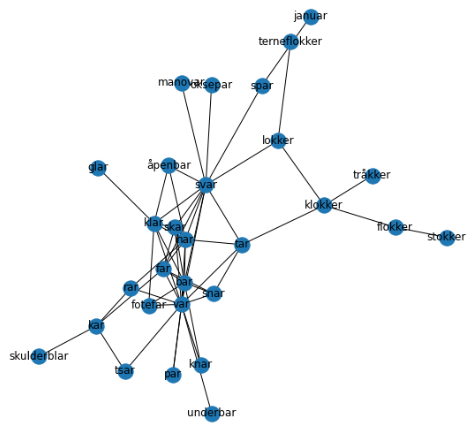


Figure A.40: Connected component *E*

A.13 Connected component *D*

In Figure A.37 we see another connected component from the annotated rhyme pair graph, here named *D*. It consists of 38 vertices.

In Figure A.38 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 13 vertices in total. The components contain words ending in oːɾɛɾ (top), øːd (left) and øː (right). 25 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.39 we see the result of manually separating the graph in Figure A.37. 4 edges were removed from the original graph, resulting in 4 connected components of size >1. The components contain words ending in oːɾɛɾ (top), øːt (left), øːd (center) and øː (bottom). No vertices were separated into single-vertex components.

2 of the edges removed were due to a wrongly annotated rhyme scheme, connecting the word "skjød" with the words "bårer" and "tårer". One edge connected the phonetically similar "glød" (gløːd) and "brøt" (bɾøːt). One removed edge contained the word "brød", usually pronounced bɾøː, sometimes bɾøːd.

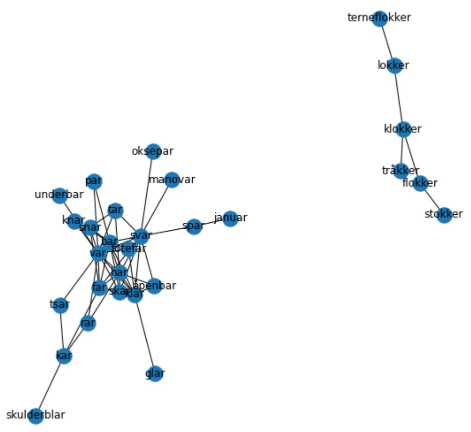


Figure A.41: Manual clustering connected component E

A.14 Connected component E

In Figure A.40 we see another connected component from the annotated rhyme pair graph, here named E. It consists of 29 vertices. In Figure A.41 we see the result of manually separating the graph in Figure A.40. 2 edges were removed from the original graph, resulting in 2 connected components of size >1. The components contain words ending in aːr (left) and okːɛr (right). No vertices were separated into single-vertex components

When running E through the HCS algorithm, it was not separated into smaller connected components.

A.15 Connected component F

In Figure A.42 we see another connected component from the annotated rhyme pair graph, named F. It consists of 29 vertices. In Figure A.43 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 9 vertices in total. 20 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.44 we see the result of manually separating the graph in Figure A.42. 4 edges were removed and one was added, resulting in 2 connected components of size >1. All removed edges were due to a mistake during rhyme scheme annotation. The added edge was to connect a separated vertex to its

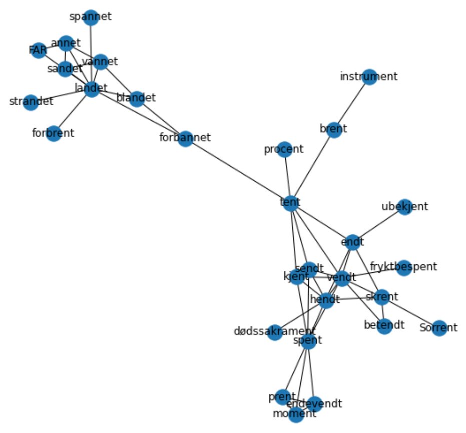


Figure A.42: Connected component F

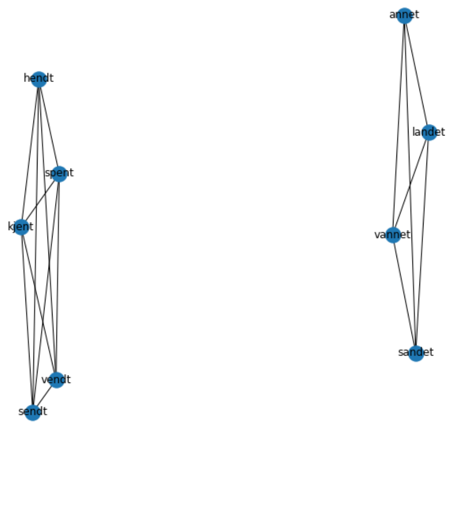


Figure A.43: Post-clustering connected component F

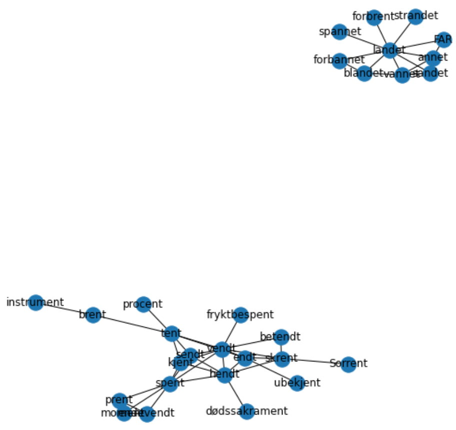


Figure A.44: Manual clustering connected component F

cluster. One vertex, the word "FAR", was separated into a single-vertex component. This word comes from the title of a poem, a line that should have been annotated with T (title) and be separated from the rest of the stanza.

A.16 Connected component G

In Figure A.45 we see another connected component from the annotated rhyme pair graph, here named G. It consists of 28 vertices.

In Figure A.47 we see the result of manually separating the graph in Figure A.45. 3 edges were removed from the original graph, resulting in 2 connected components. The connected components contain words ending in ʊnːer (top) and ʊnːe (bottom). No vertices were separated into single-vertex components

In Figure A.46 we see the result of running this through the HCS clustering algorithm. The graph was separated into 3 connected components of size >1, consisting of 10 vertices in total. 18 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs. The 3 components contain words ending in ʊnːe (left), ʊnːer (right) and both ʊnːe and ʊnːer (top).

A.17 Connected component H

In Figure A.48 we see another connected component from the annotated rhyme pair graph, named H. It consists of 27 vertices. In Figure A.49 we see the

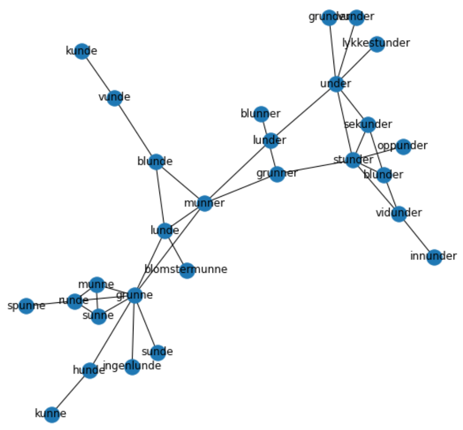


Figure A.45: Connected component G

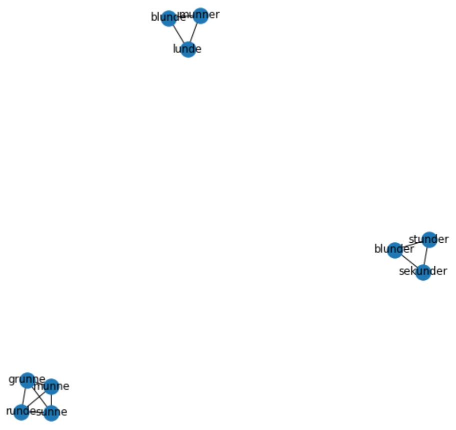


Figure A.46: Post-clustering connected component G

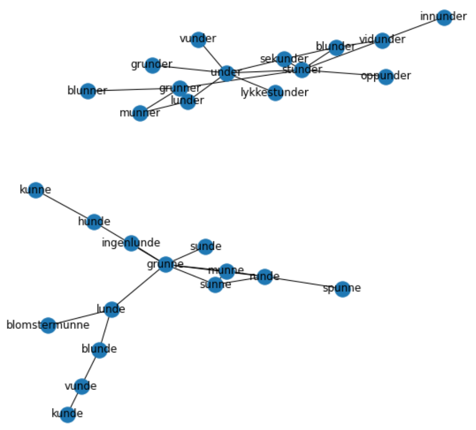


Figure A.47: Manual clustering connected component G

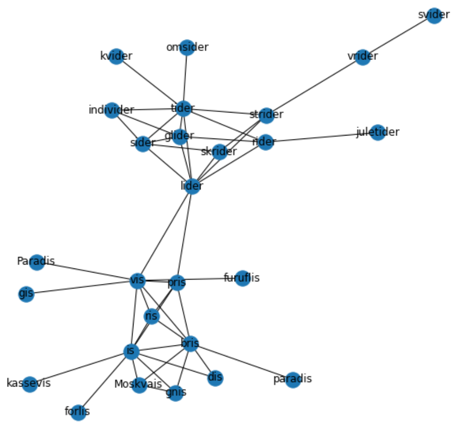


Figure A.48: Connected component H

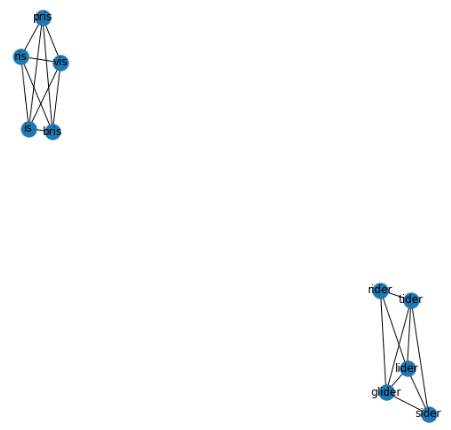


Figure A.49: Post-clustering connected component H

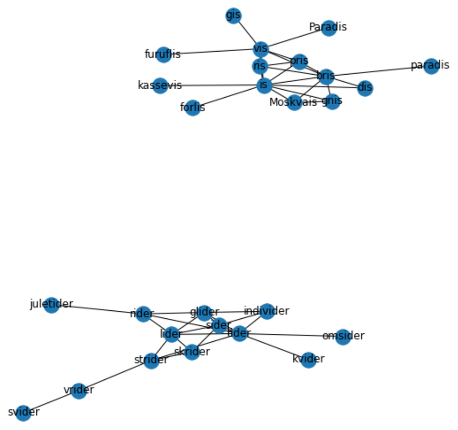


Figure A.50: Manual clustering connected component H

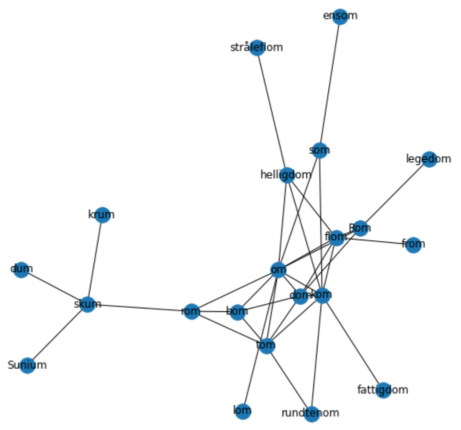


Figure A.51: Connected component *I*

result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 10 vertices in total. 17 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.50 we see the result of manually separating the graph in Figure A.48. 2 edges were removed from the original graph, resulting in 2 connected components of size >1. The components contain words that end with iːs (top) and iːdeɾ (bottom). The two removed edges were ("pris", "lider") and ("vis", "lider"), which were results of wrongly annotated rhyme schemes. No vertices were separated into single-vertex components.

A.18 Connected component *I*

In Figure A.51 we see another connected component from the annotated rhyme pair graph, named *H*. It consists of 21 vertices.

In Figure A.52 we see the result of manually separating the graph in Figure A.51. 7 edges were removed from the original graph, resulting in 2 connected components of size >1. The components consist of words ending in um (left) and om (right). Of the removed edges, 6 contain the words "tom" or "om", both which can be pronounced with both word endings. No vertices were separated into single-vertex components.

When running *H* through the HCS algorithm, it was not separated into smaller connected components.

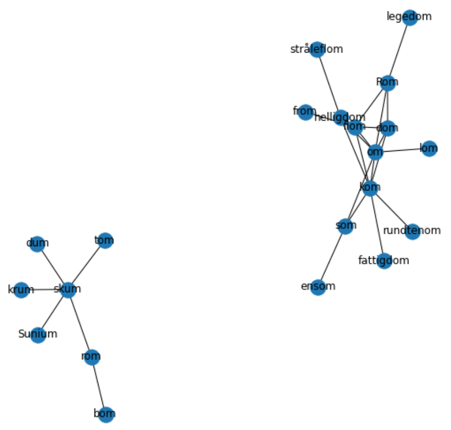


Figure A.52: Manual clustering connected component *I*

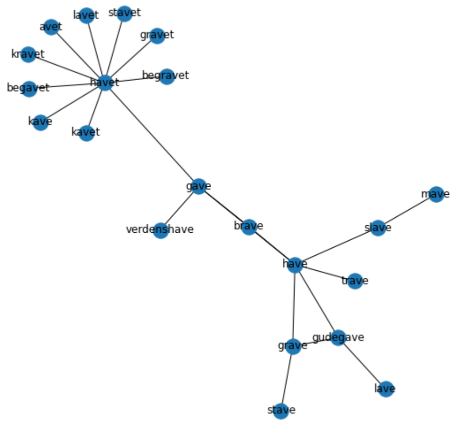


Figure A.53: Connected component *J*

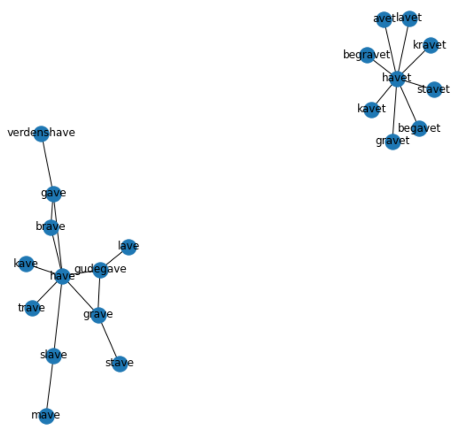


Figure A.54: Manual clustering connected component J

A.19 Connected component J

In Figure A.53 we see another connected component from the annotated rhyme pair graph, here named J. It consists of 21 vertices.

In Figure A.54 we see the result of manually separating the graph in Figure A.53. 2 edges were removed from the graph, and one was added, resulting in 2 connected components. The components consist of words ending in aːvɛ (left) and avːɛt (right). No vertices were separated into single-vertex components.

When running J through the HCS algorith, it was not separated into smaller connected components.

A.20 Connected component K

In Figure A.55 we see another connected component from the annotated rhyme pair graph, here named K. It consists of 20 vertices.

In Figure A.56 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 8 vertices in total. 12 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.57 we see the result of manually separating the graph in Figure A.55. 4 edges were removed from the original graph, resulting in 2 connected components. The components consist of words ending in amːɛn (left) and amːɛ

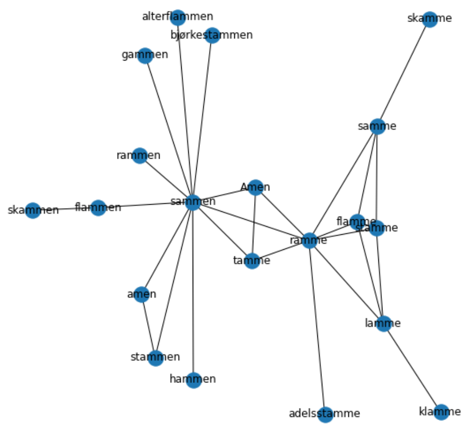


Figure A.55: Connected component K

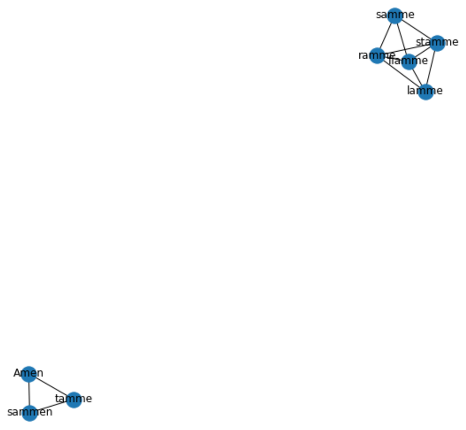


Figure A.56: Post-clustering connected component K (single vertices omitted)

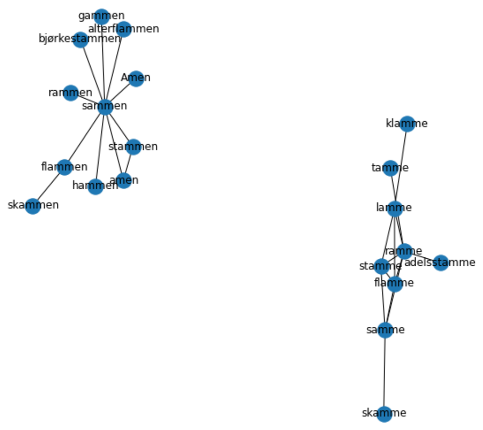


Figure A.57: Manual clustering connected component K

(right). No vertices were separated into single-vertex components.

A.21 Connected component L

In Figure A.58 we see another connected component from the annotated rhyme pair graph, named L. It consists of 19 vertices. In Figure A.59 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 8 vertices in total. 11 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs.

In Figure A.60 we see the result of manually separating the graph in Figure A.58. The edge ("vunnet", "grunnen") was removed, resulting in 2 connected components. The components consist of words ending in ʉnːən (left) and ʉnːət (right). No vertices were separated into single-vertex components.

A.22 Connected component M

In Figure A.61 we see another connected component from the annotated rhyme pair graph, here named M. It consists of 19 vertices.

In Figure A.62 we see the result of manually separating the graph in Figure A.61. 3 edges were removed from the original graph, resulting in 2 connected components of size >1. The components consist of words ending in atːer (left) and atːe (right). No vertices were separated into single-vertex components.

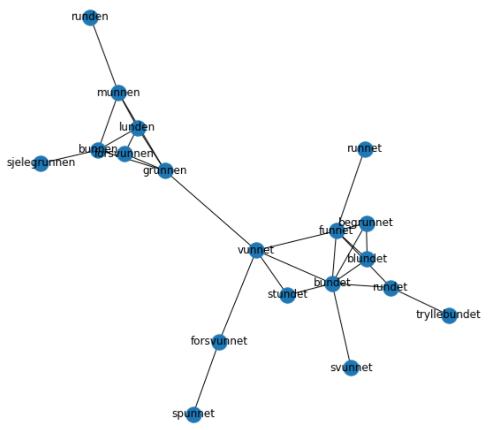


Figure A.58: Connected component L

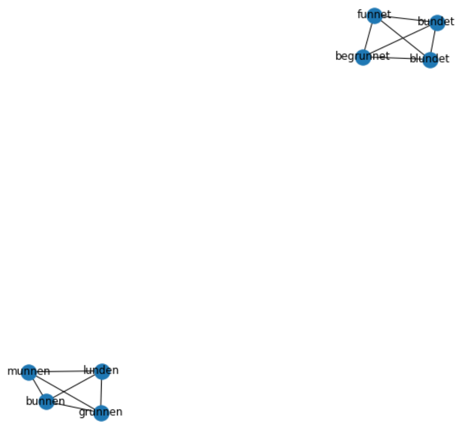


Figure A.59: Post-clustering connected component L (single vertices omitted)

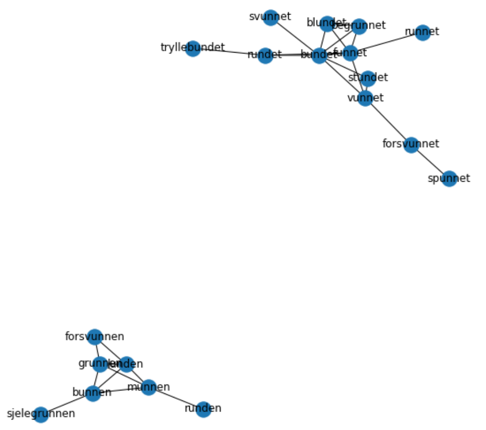


Figure A.60: Manual clustering connected component L

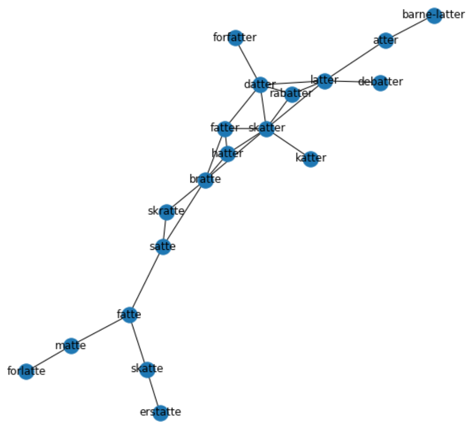


Figure A.61: Connected component M

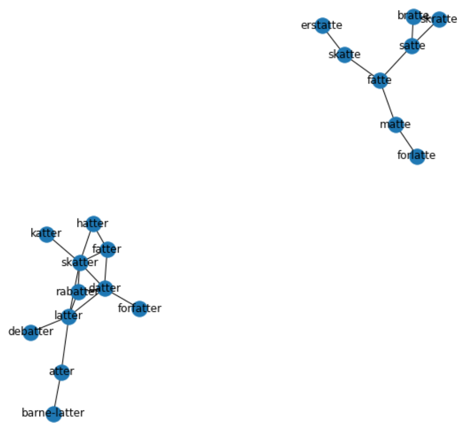


Figure A.62: Manual clustering connected component M

When running M through the HCS algorithm, it was not separated into smaller connected components.

Graph 22 was separated into 2 components. It originally had 18 vertices. Now it has 7.

A.23 Connected component N

In Figure A.63 we see another connected component from the annotated rhyme pair graph, here named N. It consists of 18 vertices.

In Figure A.65 we see the result of manually separating the graph in Figure A.63. 3 edges were removed from the original graph, resulting in 3 connected components of size >1. We see that the graph has been split into clusters containing words that end with olk, ene and ense. Due to mistakes during rhyme scheme annotation, the graph containted the edges/rhyme pairs ("tolk", "tjene"), ("folk", "ene") and ("stene", "ense"), which were removed. No vertices were separated into single-vertex components.

In Figure A.64 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 7 vertices in total. 11 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs. The ense-cluster was not extracted with the HCS algorithm.

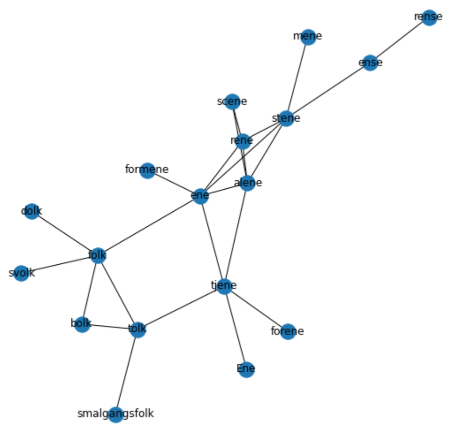


Figure A.63: Connected component N

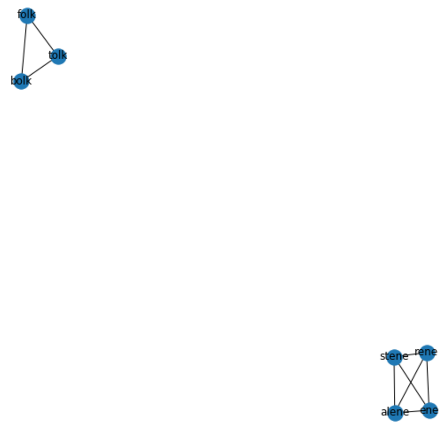


Figure A.64: Post-clustering connected component N (single vertices omitted)

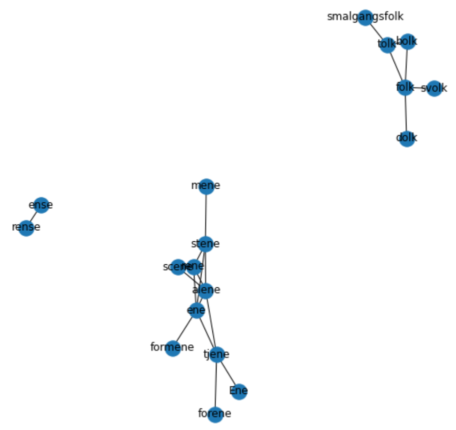


Figure A.65: Manual clustering connected component *N*

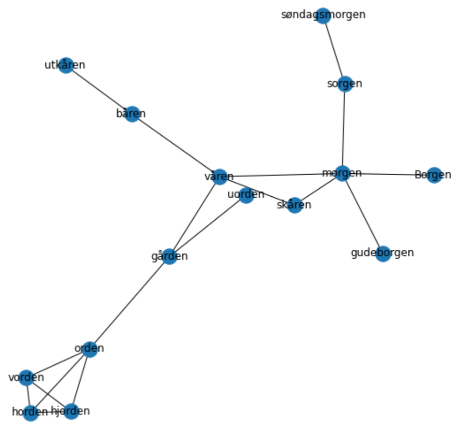


Figure A.66: Connected component *O*

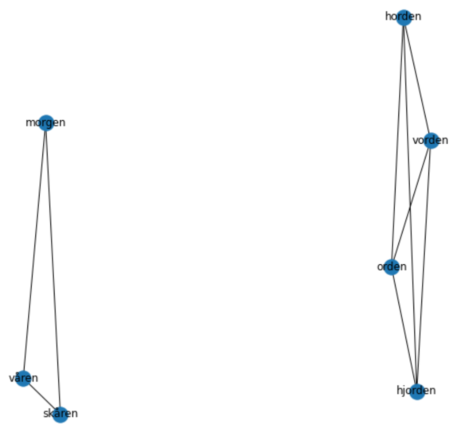


Figure A.67: Post-clustering connected component O (single vertices omitted)

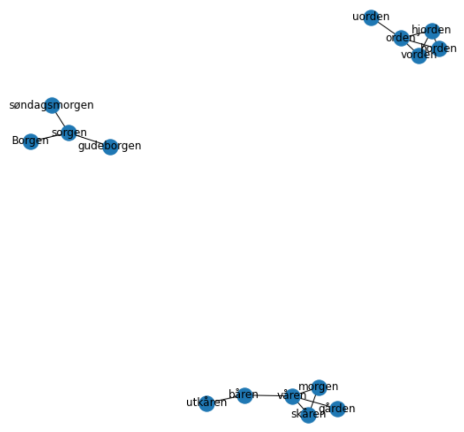


Figure A.68: Manual clustering bad connected component O

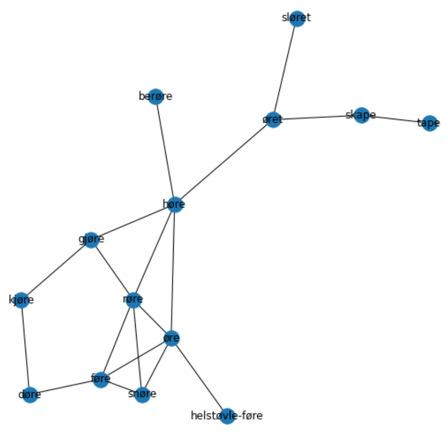


Figure A.69: Connected component P

A.24 Connected component O

In Figure A.66 we see another connected component from the annotated rhyme pair graph, here named O. It consists of 15 vertices.

In Figure A.68 we see the result of manually separating the graph in Figure A.66. 6 edges were removed from the original graph and 3 were added, resulting in 3 connected components of size >1. No vertices were separated into single-vertex components. The 3 components contain words that end with ɔrgen (left), ɔrden (right) and ɔːɔɳ̩ (bottom). These three were connected in Figure A.66 because "gården" 'the farm" and "morgen" 'morning' have alternative pronunciations. "Gården" is usually pronounced gɔːɔɳ̩, but can also be pronounced gɔrden. "Morgen" is usually pronounced mɔːɔɳ̩, but can also be pronounced mɔrgen. Three edges were added to reconnect isolated vertices to their respective clusters.

In Figure A.67 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 7 vertices in total. 8 vertices were separated into single-vertex components (not displayed in the figure), and could thus not be used to create more rhyme pairs. The ɔrgen-cluster is missing from the HCS-clustered graph, only the ɔrden- and ɔːɔɳ̩-clusters are extracted.

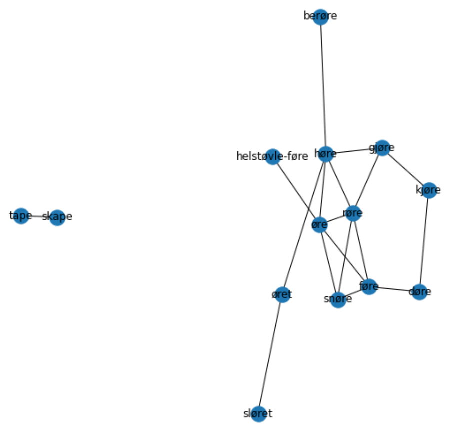


Figure A.70: Manual clustering bad connected component P

A.25 Connected component P

In Figure A.69 we see another connected component from the annotated rhyme pair graph, here named P. It consists of 14 vertices.

In Figure A.70 we see the result of manually separating the graph in Figure A.69. One edges were removed from the original graph, resulting in 2 connected components of size >1. No vertices were separated into single-vertex components. The edge that was removed is ("øret", "skape"), which connected the clusters of words ending with øre and ape.

When running the graph in Figure A.69 through the HCS algorithm, it was not separated into smaller connected components.

A.26 Connected component Q

In Figure A.71 we see another connected component from the annotated rhyme pair graph, here named Q. It consists of 13 vertices. When running this through the HCS algorithm, it was not separated into smaller connected components.

In Figure A.72 we see the result of manually separating the graph in Figure A.71. One edge was removed from the original graph, resulting in 2 connected components of size >1. No vertices were separated into single-vertex components. The edge that was removed is ("rører", "fødes"), which connected the clusters of words ending with ører and ødes.

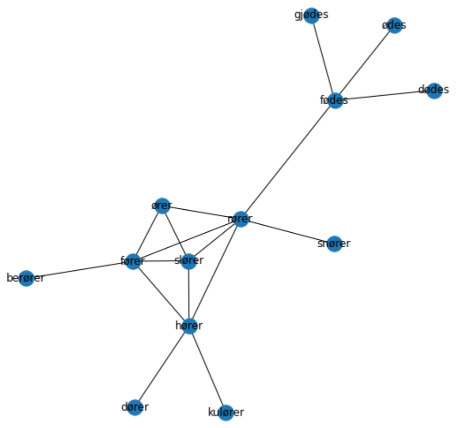


Figure A.71: Connected component Q

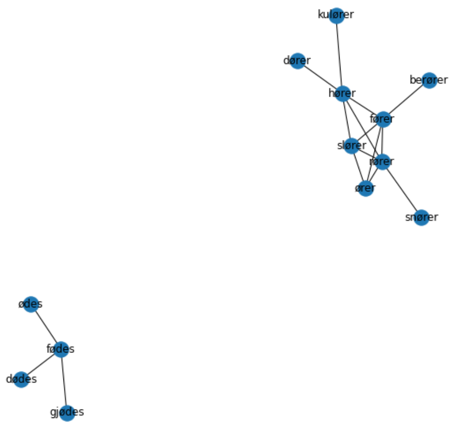


Figure A.72: Manual clustering of connected component Q

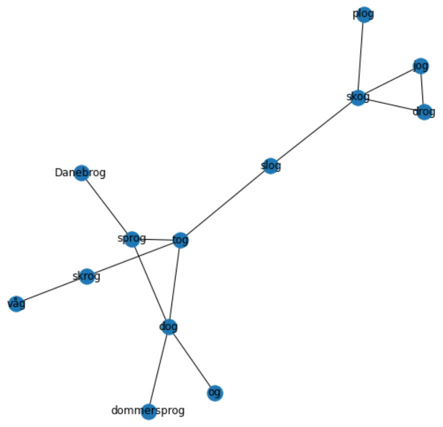


Figure A.73: Connected component R

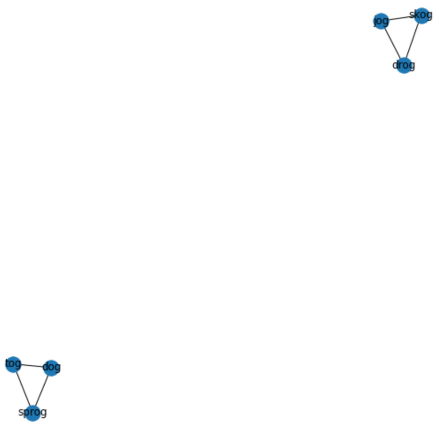


Figure A.74: Post-clustering connected component R (single vertices omitted)

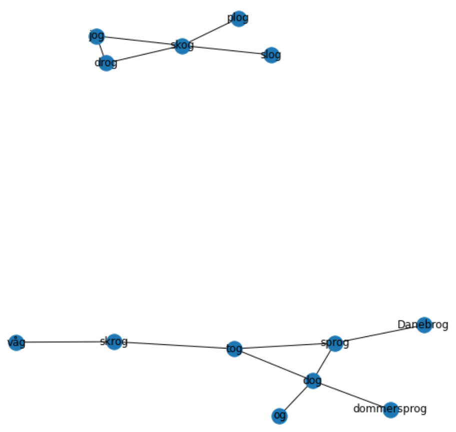


Figure A.75: Manual clustering connected component R

A.27 Connected component R

In Figure A.73 we see the last connected component from the annotated rhyme pair graph, here named R. It consists of 13 vertices. In Figure A.74 we see the result of running this through the HCS clustering algorithm. The graph was separated into 2 connected components of size >1, consisting of 6 vertices in total. 7 vertices were separated into single-vertex components, and could thus not be used to create more rhyme pairs.

In Figure A.75 we see the result of manually separating the graph in Figure A.73. One edge was removed from the original graph, resulting in 2 connected components of size >1. No vertices were separated into single-vertex components, so all 13 vertices were kept in the manual clustering.

The removed edge is ("tog", "slog"). The letter sequence "tog" can produce the heteronyms toːg 'train' and tuːg 'took' (old/Danish way to write "tok" 'took'). In the rhyme pair graph in Figure A.73 we see that this edge connects words ending in oːg (to the left) and uːg (to the left). As the "tog"-vertex has one edge connecting it to the uːg cluster, and three connecting it to the oːg-cluster, the single edge connecting the two clusters was removed.

Appendix B

Validation set results for rhyme detection

B.1 Validation set

Below are the results from the models tested on the validation set.

```
Model 1
model_name: rhyme_model_1
Val set accuracy
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       724
           1       0.96      0.92      0.94       724

    accuracy                           0.94      1448
   macro avg       0.94      0.94      0.94      1448
weighted avg       0.94      0.94      0.94      1448

---
Commutative model 1
model_name: rhyme_model_1_symmetric
Val set accuracy
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       724
           1       0.94      0.95      0.94       724

    accuracy                           0.94      1448
   macro avg       0.94      0.94      0.94      1448
weighted avg       0.94      0.94      0.94      1448

---
Model 1.2
model_name: rhyme_model_23_ratio
```

```
Val set accuracy
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       724
           1       0.96      0.93      0.94       724

    accuracy                           0.95      1448
   macro avg       0.95      0.95      0.95      1448
weighted avg       0.95      0.95      0.95      1448

---
Commutative model 1.2
model_name: rhyme_model_23_ratio_symmetric
Val set accuracy
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       724
           1       0.96      0.94      0.95       724

    accuracy                           0.95      1448
   macro avg       0.95      0.95      0.95      1448
weighted avg       0.95      0.95      0.95      1448

---
Model 2
model_name: rhyme_model_40k
Val set accuracy
              precision    recall  f1-score   support

           0       0.96      0.91      0.93       724
           1       0.91      0.96      0.94       724

    accuracy                           0.93      1448
   macro avg       0.94      0.93      0.93      1448
weighted avg       0.94      0.93      0.93      1448

---
Commutative model 2
model_name: rhyme_model_40k_symmetric
Val set accuracy
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       724
           1       0.95      0.96      0.95       724

    accuracy                           0.95      1448
   macro avg       0.95      0.95      0.95      1448
weighted avg       0.95      0.95      0.95      1448

---
```

```
Model 3
model_name: rhyme_model_dw_40k
Val set accuracy
              precision    recall  f1-score   support

           0       0.91      0.61      0.73       724
           1       0.71      0.94      0.81       724

    accuracy                           0.77      1448
   macro avg       0.81      0.77      0.77      1448
weighted avg       0.81      0.77      0.77      1448

---
Commutative model 3
model_name: rhyme_model_dw_40k_symmetric
Val set accuracy
              precision    recall  f1-score   support

           0       0.93      0.68      0.78       724
           1       0.75      0.95      0.84       724

    accuracy                           0.81      1448
   macro avg       0.84      0.81      0.81      1448
weighted avg       0.84      0.81      0.81      1448

---
Model 3.1
model_name: rhyme_model_dw_300k
Val set accuracy
              precision    recall  f1-score   support

           0       0.93      0.46      0.61       724
           1       0.64      0.97      0.77       724

    accuracy                           0.71      1448
   macro avg       0.79      0.71      0.69      1448
weighted avg       0.79      0.71      0.69      1448

---
Commutative model 3.1
model_name: rhyme_model_dw_300k_symmetric
Val set accuracy
              precision    recall  f1-score   support

           0       0.97      0.40      0.57       724
           1       0.62      0.99      0.76       724

    accuracy                           0.70      1448
   macro avg       0.80      0.70      0.67      1448
weighted avg       0.80      0.70      0.67      1448
```

B.2 Mirrored validation set

Below are the results from the models tested on the mirrored validation set.

```
Model 1
model_name: rhyme_model_1
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       724
           1       0.96      0.91      0.93       724

    accuracy                           0.93      1448
   macro avg       0.94      0.93      0.93      1448
weighted avg       0.94      0.93      0.93      1448
---

Model 1.2
model_name: rhyme_model_23_ratio
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       724
           1       0.95      0.94      0.94       724

    accuracy                           0.94      1448
   macro avg       0.94      0.94      0.94      1448
weighted avg       0.94      0.94      0.94      1448
---

Model 2
model_name: rhyme_model_40k
              precision    recall  f1-score   support

           0       0.96      0.89      0.92       724
           1       0.89      0.96      0.92       724

    accuracy                           0.92      1448
   macro avg       0.92      0.92      0.92      1448
weighted avg       0.92      0.92      0.92      1448
---

Model 3
model_name: rhyme_model_dw_40k
              precision    recall  f1-score   support

           0       0.90      0.63      0.74       724
           1       0.72      0.93      0.81       724
    accuracy                           0.78      1448
   macro avg       0.81      0.78      0.78      1448
weighted avg       0.81      0.78      0.78      1448
---

Model 3.1
model_name: rhyme_model_dw_300k
              precision    recall  f1-score   support

           0       0.94      0.43      0.59       724
           1       0.63      0.97      0.76       724

    accuracy                           0.70      1448
   macro avg       0.78      0.70      0.68      1448
weighted avg       0.78      0.70      0.68      1448
---
```

B.3 Wiktionary test set

Below are the results from the models tested on the wiktionary test set.

```
Model: rhyme_model_1
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.78      0.88      0.83      4894
           1       0.86      0.76      0.80      4894

    accuracy                           0.82      9788
   macro avg       0.82      0.82      0.82      9788
weighted avg       0.82      0.82      0.82      9788

---
Model: rhyme_model_1_symmetric
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.85      0.87      0.86      4894
           1       0.87      0.85      0.86      4894

    accuracy                           0.86      9788
   macro avg       0.86      0.86      0.86      9788
weighted avg       0.86      0.86      0.86      9788
```

```
---
Model: rhyme_model_23_ratio
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.81      0.86      0.84      4894
           1       0.85      0.80      0.83      4894

    accuracy                           0.83      9788
   macro avg       0.83      0.83      0.83      9788
weighted avg       0.83      0.83      0.83      9788

---
Model: rhyme_model_23_ratio_symmetric
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      4894
           1       0.90      0.82      0.86      4894

    accuracy                           0.87      9788
   macro avg       0.87      0.87      0.87      9788
weighted avg       0.87      0.87      0.87      9788

---
Model: rhyme_model_40k
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.87      0.77      0.82      4894
           1       0.80      0.88      0.84      4894

    accuracy                           0.83      9788
   macro avg       0.83      0.83      0.83      9788
weighted avg       0.83      0.83      0.83      9788

---
Model: rhyme_model_40k_symmetric
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.92      0.77      0.84      4894
           1       0.80      0.93      0.86      4894

    accuracy                           0.85      9788
   macro avg       0.86      0.85      0.85      9788
weighted avg       0.86      0.85      0.85      9788

---
```

```
Model: rhyme_model_dw_40k
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      4894
           1       0.95      0.94      0.94      4894

    accuracy                           0.94      9788
   macro avg       0.94      0.94      0.94      9788
weighted avg       0.94      0.94      0.94      9788

---
Model: rhyme_model_dw_40k_symmetric
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      4894
           1       0.95      0.97      0.96      4894

    accuracy                           0.96      9788
   macro avg       0.96      0.96      0.96      9788
weighted avg       0.96      0.96      0.96      9788

---
Model: rhyme_model_dw_300k
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      4894
           1       0.98      0.98      0.98      4894

    accuracy                           0.98      9788
   macro avg       0.98      0.98      0.98      9788
weighted avg       0.98      0.98      0.98      9788

---
Model: rhyme_model_dw_300k_symmetric
Wiktionary set accuracy
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      4894
           1       0.97      1.00      0.98      4894

    accuracy                           0.98      9788
   macro avg       0.98      0.98      0.98      9788
weighted avg       0.98      0.98      0.98      9788

---
```

Appendix C

Evaluation instructions for human evaluators

The instructions were given in Norwegian. The original Norwegian instructions, and the English translation are given below:

C.1 Norwegian instruction

```
Evaluering av AI-genererte vers og rim
Oppgaven din er å bruke din intuisjon til å finne ut hvilke vers som er skrevet av
mennesker, og hvilke som er skrevet av AI. Du blir også bedt om å vurdere i
hvilken grad versene rimer.

Med rim menes enderim, som for eksempel:

Jeg liker vann
Hjerter i brann

Vi bruker en skala fra 0 til 3, der 0 betyr at verset ikke inneholder enderim,
og 3 betyr at det er tydelig at verset  er skrevet på rim.

1 brukes om noen linjer i verset nesten rimer, som for eksempel:

Jeg vil prate
Dere er smarte

2 brukes om det rimer noe, for eksempel at to linjer i diktet rimer men ellers ikke.

Først er det 15 spørsmål der du får se 2 vers side om side. Her er ett vers
AI-generert og ett menneskeskrevet.
Deretter kommer det 20 spørsmål hvor du vurderer ett og ett vers uavhengig.

Tusen takk og lykke til!

NB: Versene brukt er skrevet fra år 1900 til i dag. Det er mulig at noen
```

```
vers inneholder ordbruk som reflekterer politikk/holdninger som ikke er gjengs i dag.
```

C.2 English translation

```
Evaluation of AI-generated and rhyme
Your task is to use your intuition to find out what stanzas are written by humans,
and what are written by AI. You are also asked to evaluate to what
degree the stanzas rhyme.

With rhyme, we mean end-rhyme, as for example:
Jeg liker vann (eng: I like water)
Hjerter i brann (eng: Hearts on fire)

We use a scale from 0 to 3, where 0 means that the stanza does not contain end-rhyme,
and 3 means that it is clear that the verse is written in rhyme.

1 is used if some lines in the verse almost rhyme, as for example:
Jeg vil prate (eng: I want to talk)
Dere er smarte (eng: You are smart)

2 is used if it rhymes some, for example if only two lines in the stanza
rhyme but not the other.

First, there are 15 questions where you will see 2 stanzas side by side.
Here one stanza is AI-generated and one is written by humans.
After that, there are 20 questions where you evaluate one stanza at the time.

Thank you and good luck!

NB: The stanzas used are written from year 1900 to today.
There is a possibility that some stanzas contain words
that reflect politics/attitudes that are not valid today.
```