Markov-Based Lyric Generation Using Computed Rhyme Templates

Marissa Graham[1], Reed Perkins[2]

[1]Mathematics Department, [2]Computer Science Department

Brigham Young University

Provo, UT 84602 USA

marissa.graham101@gmail.com, reedperkins32@gmail.com

Abstract

Matching text to a certain rhyme pattern is an important part of effective song lyric generation, especially in the case of rap, but most current lyric generation systems are limited to a user-defined rhyme template. We have developed a system that extracts relevant rhyme features as well as rhythm from a given template text, and then uses a Markov model to generate a novel text in the style of a given corpus, which matches the rhyme and metric structure of the original template.

Introduction

Making words rhyme for a living is one of the great joys of my life... That's a superpower I've been very conscious of developing. I started at the same level as everybody else, and then I just listened to more music and talked to myself until it was an actual superpower I could pull out on special occasions.

- Lin-Manuel Miranda

The problem of fitting text to a rhythmic template is a well-established one in the field of computational creativity, with many systems also able to fit simple rhyme constraints. Contemporary approaches can be roughly categorized as those which are Markov-based and those which are template based. (Oliveira 2009), (Papadopoulos, Roy, and Pachet 2014), and (Barbieri et al. 2012) all discuss the Markov approach, with the goal of fitting longer-range constraints by building them into the Markov model itself.

A Markov-based approach is well suited for basic text generation systems that seek to emulate the style of an existing corpus. However, the metrical or rhyme patterns that are central to song lyrics, rap or otherwise, are difficult to implement with the short-range scale of a simple Markov model. Since the constraints in the more free-form structure of rap lyrics are less universal, we use a generate-and-test approach instead of building constraints into the Markov model itself.

The systems discussed in (Oliveira 2009), (Özbal, Pighin, and Strapparava 2013), and (Toivanen et al. 2012) take a template-based approach, where the structure of a valid sentence is trained on a large corpus of existing poetry (Oliveira 2009), or by replacing pieces of an existing poem until a reasonably novel result is obtained (Toivanen et al. 2012). We generalize on this concept by extracting only the structure of an existing poem, so the pieces are replaced all at once. This sacrifices some coherency at the expense of high novelty.

In order to extract only the structure of an existing text, we need to parse both the rhythm and the rhyme pattern. Fortunately, decoding rhythm is relatively straightforward. (Barbieri et al. 2012) outline a method to extract rhythmic information from stress markers found in phonetic dictionaries. We simply extract the stress information from the vowel phonemes in a given word, and concatenate the results.

Rhyme, however, is far more difficult concept to implement. Not only do sounds not share a one-to-one relationship with letters in the alphabet, but imperfect rhyme cannot be modeled with a binary approach. (Bay and Bodily 2017) explore a system to evaluate rhyme through the use of genetically learned weights for linguistic features of vowels and consonants, building on the work of (Hirjee and Brown 2010), who describe how an alignment matrix of phonemes can be created and statistically analyzed to find rhyme patterns. We use the results of these authors in the construction of our own rhyme evaluator.

On top of the difficulty in checking whether words rhyme, we also face the difficult task of determining which words need to rhyme. In most existing systems, the rhyme scheme to be imitated must be user-determined, and is limited to rhymes occurring at the end of a line (Barbieri et al. 2012).

Since rap and hip hop music are characterized by a much wider variety of rhyme features and rhythmic schemes (Hirjee and Brown 2010), this is not a feasible strategy. In order to automatically generate rap lyrics, we need a system that can determine for itself which words need to rhyme. We do so by choosing the strongest rhyme pairs in a template text, and mimicking them in our output.

Approach

Template Generation

We use the CMU Pronouncing dictionary to parse plaintext words into sequences of ARPAbet phonemes, where vowel phonemes are tagged with their lexical stress. For example, the word "forgotten" is parsed into the phoneme sequence "F ER0 G AA1 T AH0 N".

The rhythm of the word is therefore "[0, 1, 0]", that is, a stressed syllable preceded and followed by unstressed syllables.

We used the CMU LOGIOS Lexicon tool to retrieve approximate phonetic information for words not found in the main CMU dictionary, which is a frequent occurrence for rap lyrics. Since this tool does not report lexical stresses, we defined all vowel phonemes for unknown words to be unstressed.

To extract the rhythm of a template text, we simply concatenate the rhythm sequence for each word into a single list, while storing the necessary translation information to convert between location indices in the sequence of words and the sequence of syllables for the template. We also store the desired breakpoints for the template.

These breakpoints are simply the syllable locations in the pattern that cannot occur in the middle of a word as we fill the template, and keep us from having to stretch words out unnaturally when reading the final result.

To mimic the original most closely, we can insert a breakpoint at the end of every word, but this is a stronger constraint than necessary to get a rhythm that sounds like the original. Instead, we can also insert a breakpoint at the end of each phrase, which we approximate by looking at line breaks and other significant punctuation. This largely preserves the metric structure of the original, while giving us more flexibility as we fill the template.

For example, the template text

buddy you're a boy, make a big noise, playing in the street, gonna be a big man some day, you got mud on your face, you big disgrace, kicking your can all over the place

has the stress pattern

```
# / # / #, # / # #, # / / / #, # / #
/ # # # #, # # # # # # #, # # / #, # /
# # # # / / #,
```

where we use a "#" character to denote a stressed syllable, a "/" character to denote an unstressed syllable, and commas to denote the phrase breakpoints.

Rhyme Template We would also like to mimic the rhyme scheme of a given text. In the absence of a human ear to determine which rhymes are important to making the text sound lyrical, we approximate how important the rhymes are by how closely they match. We do so by scoring all potential rhymes between pairs of local syllables in the template text, and then taking the top few pairs as those that need to be matched.

Rhyme Scoring Computationally evaluating the strength of a rhyme is not a trivial problem. Previous approaches such as (Bay and Bodily 2017), (Hirjee and Brown 2010), and (Hinton and Eastwood 2015) have taken an approach based on scoring the match between the individual phonemes in a pair of words to be evaluated. (Hirjee and Brown 2010) do so by analyzing phoneme frequency in a given corpus, while (Hinton and Eastwood 2015) do a binary comparison of the phonemes involved; that is, syllables rhyme if they have the same vowel sound and a similar suffix. We use the results of (Bay and Bodily 2017) for our method, which takes these concepts further by training on a large rhyme database in order to create an evaluation function which scores individual phonemes based on their IPA linguistic features.

Table 1: Rhyme Score Alignment Matrix

-	IH1	OW0	AH0	AH0	AA1	ER0
IH0	0	0.0156	0	0	0	0.3314
AA1	0.4544	0	0	0	0.8345	0
IH0	0	0.0406	0	0	0	0.3486
IH0	0	0	0	0	0	0
AA1	0.489	0	0	0	0.9628	0
ER0	0	0.3292	0	0	0	1

The results of (Bay and Bodily 2017) take the form of optimal weights to score each of these linguistic features, so we stored these as well as the linguistic features for each ARPAbet phoneme in order to calculate a rhyme score between any syllable of a word and any syllable of another word. We deliberately give a score of zero to any pair including a word with minor rhyme significance such as "a", "I", or "the", in order to cut down on the false positives in our results. We also give a zero score to identical syllables, in order to avoid having our system rhyme words with themselves.

Rhyme Matrix Once we have the ability to score rhymes between syllables, we look for strong rhymes in the template text by calculating the scores between local pairs of syllables.

To do so, we construct an $m \times m$ matrix, where m is the number of syllables in the template text. Then we fill in the entries along a superdiagonal of width 30 by computing the rhyme score for that word/syllable pair. Figure 1 illustrates the global structure of this matrix, and Table 1 demonstrates some example scores between the syllables in "impoverished, in squalor" (along the side), and "hero and a scholar" (along the top).

Next, we need to choose which rhyme pairs to match. We would like to have enough rhyme pairs to fill about 10% of the original length since this roughly matches the amount of rhyming words in a sonnet and does not result in an overly constrained template. We determine what cutoff percentile we need in order to get this number of nonzero entries, and then zero out all entries below it. In practice, this cutoff score was roughly 0.95. The (i, j) coordinates of the nonzero entries then represent a requirement that the ith and jth syllables rhyme in the generated result.

Template Filling

While a Markov model is a powerful method for generating plausible and somewhat coherent text in a given style, it did not prove to be an effective method for matching rhyme pairs. There are simply not enough options in an average sampling of follow-words to match a rhyme scheme. We therefore decided to fill in the rhyme pairs in the template first to get "anchor words".

Figure 1: Rhyme scoring matrix before cutoff of low scores.

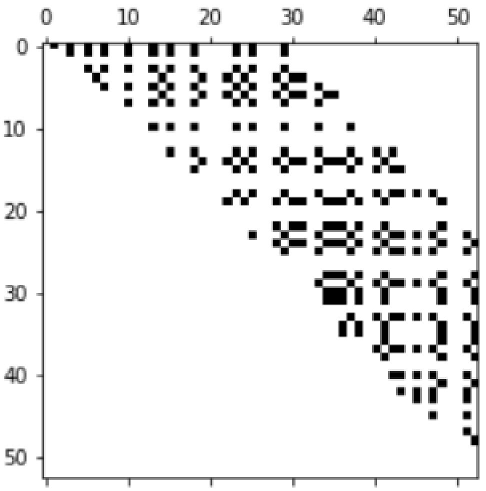
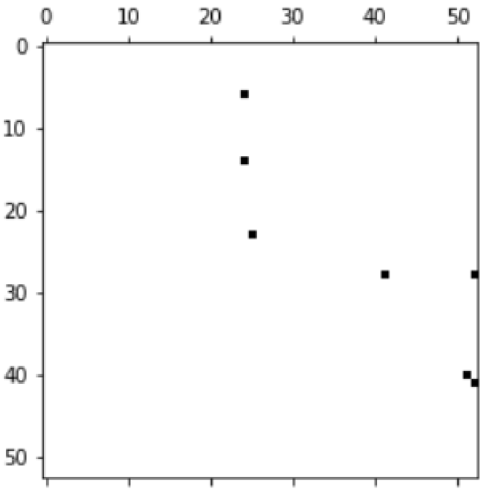


Figure 2: Rhyme scoring matrix after cutoff of low scores.



After we have these, we can then use a Markov model to fill the rest of the template according to the rhythm pattern, which is much more feasible and only occasionally results in a need to start over and choose a random word from the entire corpus.

Anchor words Choosing rhyme pairs reduces to two main steps: choosing an initial word for a rhyme pair, and matching an existing word in a rhyme pair. The generation process then simply consists of iterating through the nonzero entries of the template rhyme matrix, checking whether either syllable is already occupied, and then choosing zero, one, or two rhyming words to fill the template accordingly.

For the initial choice in a pair, we would like to favor words from the source text that are easy to rhyme. Since a matching vowel phoneme is by far the most important feature in a high rhyme score, we approximate "rhymeability" by how frequently the vowel phoneme occurs in the source text, and sample from the resulting probability distribution. Once we have chosen a phoneme, we randomly sample from the words in the source text which contain that phoneme, and choose one which matches the rhythm of the template.

To match an existing word, we extract the vowel phoneme for the original, sample the words that contain that phoneme, and choose one that rhymes best with the original while also matching the rhythm of the template. An example of this process is shown in Figure 3.

Figure 3: Example output from the system while filling the rhyme words of a template.

```
Fill 5 rhyme pairs:

Get a match for syllables 0 and 24
Both empty, pick both words at random
Get samples using the vowel phoneme IH
Add ("if" IH1 F) and ("years" Y IH1 R Z)

Get a match for syllables 9 and 21
Both empty, pick both words at random
Get samples using the vowel phoneme AH
Add ("mother" M AH1 DH ER0) and ("orphan"
AO1 R F AH0 N)

Get a match for syllables 27 and 31
Both empty, pick both words at random
Get samples using the vowel phoneme AW
Add ("now" N AW1) and ("around" ER0 AW1 N D)

Get a match for syllables 27 and 40
Need to match rhyme with first word "face"
Get samples using the vowel phoneme EY
Add ("refrain" R IH0 F R EY1 N)

Get a match for syllables 31 and 40
Occupied:  ( # around, # ) and ( / refrain )
```

Original: buddy you're a boy, make a big noise, playing in the street, gonna be a big man some day, you got mud on your face, you big disgrace, kicking your can all over the place
Mimicked: if / # / #, # / # #, mother / / #, # / # / # # orphan, # # years # # now, # # around, # / # # # # / refrain

Markov model The rest of the words in the template are filled in using a Markov model. We work forwards and backwards from the words in the template, sampling from the columns instead of the rows in the latter case.

Once we have some words to choose from, we check how well each one fits the rhythm of the template, and discard those which run into a breakpoint, an edge, or a neighbor. We then take the top 25% of the options, and scale their score by their length as well as how "followable" they are.

In creating our followability score, we seek to avoid dead ends without only choosing common, boring words. To do so, we define followability to be the number of unique words preceding/following our current choice, scaled by the number of occurrences in the text. This way, we favor followability proportional to frequency.

Once all these scores are calculated, we again take the top

Figure 4: A sample text generated by the system, using the first song of the musical *Hamilton* as a corpus and the intro of "We Will Rock You" by Queen as a template. We show the template after filling in rhyming words, and then the overall result.

```
Corpus text:  547 words, 256 unique

Template text:  buddy you're a boy, make a
big noise, playing in the street, gonna be a
big man some day, you got mud on your face,
you big disgrace, kicking your can all over
the place

Fill in Rhyming Words:  ridden # / #, # / #
#, every / #, # / # / # # # whence, # # his
# # dead, # # / went, # / # # # # / / wait

Overall Result:  ridden two years take, your
name ing for, every treatise, on the brother
took up from whence, you to his first are
dead, or barter went, quick moved in with
him alex you wait
```

25% of our choices, and sample from the probability of their scores to get the final result. This allows us to favor the best options while still generating unique results every time.

If there are no good options available, we have the ability to compromise the stress pattern, as long as the results are of the right length. This cuts down the frequency of dead ends to a much more manageable level, and we handle any remaining dead ends by sampling from the entire corpus at random.

Joining Stubs As we fill in the "holes" in the template, we need to join the results of our forwards and backwards generation in the middle. Once we have three or less syllables to fill in the middle of a hole, the system looks for words of the correct length which follow the word on the left-hand side of the hole, and are followed by the word on the right-hand side of the hole. If there are none, it adds a word on the left as usual and repeats the process until the hole is filled.

With a small corpus, most of the time there are no good filler word available, but by doing this we can at least somewhat preserve the plausibility and flow of our generated text.

Results

Overall, we have a system that is fairly inexpensive to run which successfully extracts rhyme patterns from a template and generates text with the desired structure. The results are highly unique even with a small input corpus.

While the semantic and grammatical coherency of the results is low, similar to a basic Markov babbler, the structure of the system lends itself readily to improvement of the individual pieces and should allow significant improvements to the coherency within the framework of the overall approach.

A small example of the results generated is given in Figure 4. We also include the results of a slightly larger example in Figure 5, sourced from the song "Wait for It" from *Hamilton*.

Uniqueness

To test the novelty of the output of our system, we compared the similarity between repeated results for the same source corpus and template. For each song, we generate ten results using the first verse of that song as source text and the entire song as the corpus and compare them all to each other to get an idea of the extent to which the system produces similar results given the same input.

We track both the average similarity between repeated results for the same song, as well as the maximum similarity score. The similarity score measures the degree to which a template is a word-for-word match of another, with a 1 indicating a perfect match and a 0 indicating no matching words at corresponding syllable indices. Results are summarized in Table 2.

We used five songs (Kendrick Lamar's "Good Kid", Cardi B's "Bodak Yellow", Nicki Minaj's "Chun Li", Drake's "Nice For What", and the track "Alexander Hamilton" from the musical *Hamilton*) selected by taking the top listing for each artist on Spotify at the time of writing this paper.

Contribution and Future Work

The main contribution of this project is a system that can extract the rhyme patterns from a template text and then generate new text according to that structure. While (Hirjee and Brown 2010) are able to identify a variety of rhymes automatically, they do not use the results to generate new text. Furthermore, most text generation systems that consider rhyme are only able to match rhymes in a user-specified pattern, and only consider rhymes at the ends of words and the end of a line.

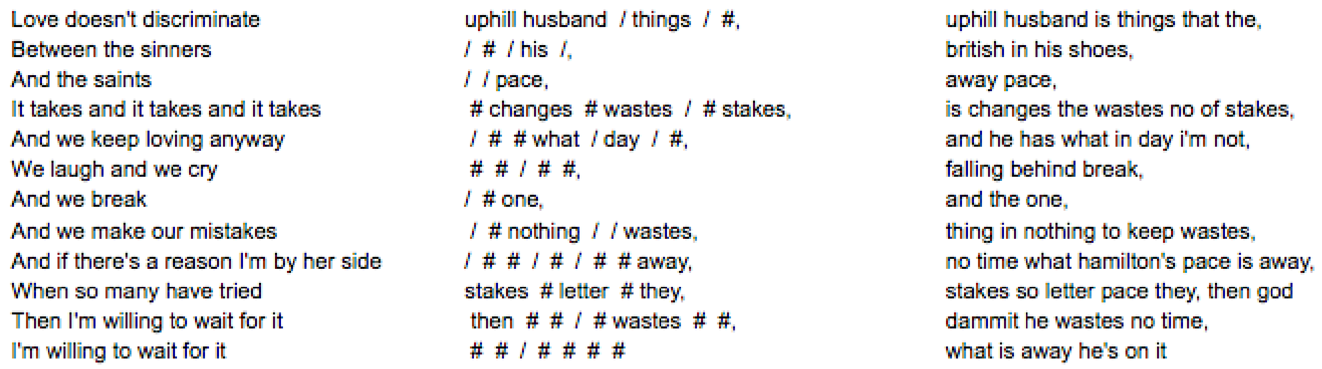
Our system, on the other hand, combines a fairly standard Markov model with a more sophisticated rhyme scorer that is not limited to end-of-line rhymes or specific syllables, and which incorporates more of the linguistic features involved, into a generalized template framework for poetry and rap generation.

Since the framework of the system is extensible enough to allow improvement of each constituent part separately, we can consider future work by discussing improvements to be made in each individual piece. The main areas of focus are overall coherency, and more sophisticated rhyme detection.

To improve the rhyme detection, we would like to allow the system to detect multisyllabic, imperfect, and slant rhymes, as done by (Hirjee and Brown 2010). To do this, we can look at small diagonals with high scores as seen in the bottom right of Figure 2 to get multi-syllable rhymes, and look at score ranges other than the highest scores in order to get imperfect rhymes. We could also look at the scoring patterns for text with human-marked rhyme words to see what other features contribute to whether a rhyme is perceived as important.

Since coherency in the results is noticeably dependent on the coherency of the anchor words, we can likely get significant improvements in our results by choosing anchor words which are semantically related, or by attempting to mimic the grammatical and semantic relationships between the corresponding original words in the template instead of just the

Figure 5: A sample text generated by the system, using the song "Wait For It" from the musical *Hamilton* as a corpus and its first verse as a template. We show the template after filling in rhyming words, and then the overall result.



Love doesn't discriminate	uphill husband / things / #,	uphill husband is things that the,
Between the sinners	/ # / his /,	british in his shoes,
And the saints	/ / pace,	away pace,
It takes and it takes and it takes	# changes # wastes / # stakes,	is changes the wastes no of stakes,
And we keep loving anyway	/ # # what / day / #,	and he has what in day i'm not,
We laugh and we cry	# # / # #,	falling behind break,
And we break	/ # one,	and the one,
And we make our mistakes	/ # nothing / / wastes,	thing in nothing to keep wastes,
And if there's a reason I'm by her side	/ # # / # / # # away,	no time what hamilton's pace is away,
When so many have tried	stakes # letter # they,	stakes so letter pace they, then god
Then I'm willing to wait for it	then # # / # wastes # #,	dammit he wastes no time,
I'm willing to wait for it	# # / # # # #	what is away he's on it

Table 2: Similarity Scores for 100 Runs of Verse Template Generation

	Kendrick Lamar	Cardi B	Nicki Minaj	Drake	Hamilton
Corpus Size (words)	510	768	447	741	561
Average Verse Length (words)	120	106	100	109	77
Average Similarity Score	0.013	0.023	0.017	0.019	0.012
Max Similarity Score	0.125	0.289	0.202	0.229	0.173

fact that they rhyme, similar to the process of the BRAINSUP system developed by (Özbal, Pighin, and Strapparava 2013).

Including part of speech or other simple grammatical constraints in the process of choosing words from the Markov distribution could also improve the results significantly.

We also can improve coherency by reducing the frequency of dead ends, where we need to sample words at random. We can do so by both implementing dead-end constraints into the Markov model itself, as discussed by (Pachet et al. 2001), or by giving the system the ability to sample new words according to correct part of speech instead of at random.

On a larger scale, we need better evaluation of the results. We already have a good metric of how well they fit a rhythmic template, but we do not have an objective evaluation of the coherency of the text, or how well the rhyme template chooses which rhymes are important. Training using feedback from online grammar checkers as well as Likert scale surveys would be the next step in this direction.

References

Barbieri, G.; Pachet, F.; Roy, P.; and Degli Esposti, M. 2012. Markov constraints for generating lyrics with style. In *ECAI*, volume 242, 115–120.

Bay, B., and Bodily, P. 2017. Dynamically scoring rhymes with phonetic features and sequence alignment.

Hinton, E., and Eastwood, J. 2015. Playing with pop culture: Writing an algorithm to analyze and visualize lyrics from the musical hamilton.

Hirjee, H., and Brown, D. G. 2010. Using automated rhyme detection to characterize rhyming style in rap music. *Empirical Musicology Review* 5(4):121–145.

Oliveira, H. 2009. Automatic generation of poetry: an overview. *Universidade de Coimbra*.

Özbal, G.; Pighin, D.; and Strapparava, C. 2013. Brainsup: Brainstorming support for creative sentence generation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 1446–1455.

Pachet, F.; Roy, P.; Barbieri, G.; and Paris, S. C. 2001. Finite-length markov processes with constraints. *transition* 6(1/3).

Papadopoulos, A.; Roy, P.; and Pachet, F. 2014. Avoiding plagiarism in markov sequence generation. In *AAAI*, 2731–2737.

Toivanen, J.; Toivonen, H.; Valitutti, A.; Gross, O.; et al. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the Third International Conference on Computational Creativity*.