CS224N Final Project
cRhyme: A Computer Engine for Generating Rhyming Sentences

Gil Shotan
Stanford University
gilsho@cs.stanford.edu

Rafael Ferrer
Stanford University
rmferrer@cs.stanford.edu

Abstract

We introduce cRhyme, a computer system that transforms pairs of sentences to make them rhyme. cRhyme takes in a pair of sentences and performs a series of syntactic transformations to one or both sentences in order to make them rhyme without altering the meaning of either. This makes cRhyme the to-go tool for amateur poets and songwriters who know what to say but have trouble expressing it in a poetic form.

1. Introduction

As any amateur poet can attest, generating a pair of rhyming sentences that convey a given meaning is an arduous task. Several services exist on the web that assist such an amateur poet in finding synonyms to words and rhyming words. However, there is no existing service that tries to tackle the full problem of paraphrasing sentences to make them rhyme end-to-end. In this paper we present the problem of taking pairs of sentences and syntactically transforming them to make them rhyme without altering their meaning. For example, we would like to transform the following pair of garbled U2 lyrics:

> "Twist of destiny and sleight of hand,
> On a bed of nails she looks me wait"

Into a pair of sentences the rhyme, such as the original form:

> "Sleight of hand and twist of fate
> On a bed of nails she looks me wait"

The task at hand involves several different sub-tasks, each daunting in its very own right. First we need to be able to accurately determine if two words rhyme or not. Then, if we cant find examples of words that rhyme we must paraphrase the sentence in order to generate alternate representations of the sentence that preserve its original meaning. Lastly, we need a means of ranking different candidates to ensure we can produce quality results. The interesting and fun part about the problem is that it gives us a lot of flexibility in producing results. As poetry is typically presented in forms rarely used in day to day speech, and often stretches the limits of the language to gain expressive power, people have come to expect poetry that lacks a well defined syntactic structure. Therefore we have a greater degree of syntactic freedom in producing our substitute sentences. Moreover, since poets often use analogies in their poems that are not straightforward for the readers to understand, we also have a lot of wiggle room in terms of the content of the structure that we produce. As long as our candidate sentences can bare some resemblance to the original sentence, a client of our system can attribute the difference to artistic freedom and claim that a complex analogy is taking place in her lyrics.

2. Related Work

Quite of bit of work has been done on the topic of word pronunciation, culminating in the widely used Carnegie Mellon Pronunciation Dictionary[7]. Several developers have used this resource as a foundation to develop rhyming engines, taking as input a word and generating words that rhyme with the aforementioned word. We found these engines satisfactory for our application, and in particular our application uses the Rhyming Dictionary 0.9 by Brian Langenberger[4]. Similarly, much work has been done on finding synonyms of a given word, most notably the WordNet project[5]. We utilize the the WordNet project in our application through a Java plugin written by Brett Spell, of the Computer Science and Engineering department at Southern Methodist University[6]. However, we've found very little work relating to sentence transformation, or paraphrasing, that would be relevant to our application. While paraphrasing sentences have many applications[1], including questions answering, text polishing in natural language generation, text simplification in computer aided reading, text summarization, and sentence similarity computation in the automatic evaluation of machine translation, none of the aforementioned applications are particularly concerned

with rearrangement of the words in the sentence, a very important property for our application. As a result, most of techniques we surveyed for dealing with the problem of paraphrasing tend to focus on the constituent level, rarely venturing to alter the deep syntactic structure of the sentence. We identified 4 techniques used in practice[8]:

1. Rule based approaches -In this approach paraphrasing is performed by subjecting the target sentence to transformations dictated by a paraphrase rule corpora.

2. Thesauras based methods - In this method candidates words are identified in the target sentence and replaced by synonyms.

3. NLG-based methods - This approach involves transforming sentences into a knowledge representation form, and then using natural language generation techniques to convert the knowledge form back into natural language form. Underlying this technique is the notion that one representation maps to many different natural language forms.

4. SMT-based methods - In this approach, also known as ”pivoting” the target sentence is converted into one or more different languages, called the pivot language, and then translated back into the original language[9]. The guiding principle is that two phrases that translate into the same phrase in the target language share the same meaning.

NLG based methods are too complex to implement for the scope of this project. Our survey of SMT based systems yielded very poor paraphrasing results on our data set, and in addition, very infrequently did we observe the type of syntactic re-arrangement necessary for our applications[2]. This left us with approaches 1 and 2 which we combine together in our system.

3. Method Overview

Recall that our goal is to take two sentences as input and perform transformations on them such that the resulting sentences rhyme and preserve their original meaning. In order to achieve this goal we first try to generate as many candidate sentences as possible, and then narrow the results down by scoring and ranking the candidate sentences produced. Our approach can be broken down into 4 steps

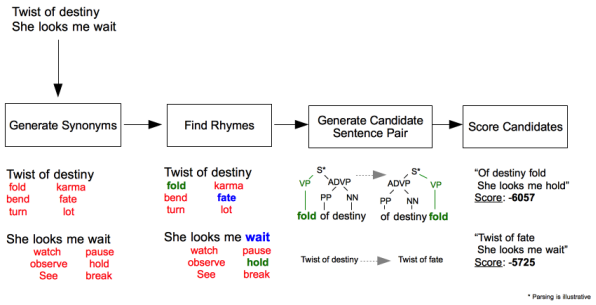

Figure 1. Illustration of Methodology

3.1. Generating Synonyms

The very first step in our approach is to populate a list of synonyms for each word in both sentences to produce a pool from which to choose rhymes. For each word we consider as synonyms the list of synset objects related to it by WordNet, but due to the relax-constrained nature of our problem we also consider the hypernyms and the hyponyms of a given word as potential synonyms.

3.2. Finding Rhymes

Following step 1, we search through every pair of words from different sentences, and for each such pair of word we search through each pair of synonyms associated with that word, including the original words themselves. If we find a pair of words that rhyme then we generate a candidate sentence pair. In order to detect if a pair of words rhyme, we use the Rhyming Dictionary 0.9 developed by Brian Langenberger, which is built on top of CMUs pronunciation dictionary[4].

3.3. Generating Candidate Sentence Pairs

Once we find a pair of rhyming words we generate a candidate sentence pair by first substituting the original words with the rhyming synonyms, if necessary, and then proceed to rearrange the sentence to place the rhyming words at the end of their respective sentences. In order to do so with minimal disturbance to the rest of the sentence, we use algorithm 1.

An illustration of the algorithm is given in figure 2. Essentially, the algorithm is trying to push the target word to the end of the sentence while trying not to disturb the unrelated parts of the sentence, thus preserving subtrees that are syntactically correct in the re-arrangement process. Furthermore, the re-arrangement performed at each step matches the kind of transformations that people, and especially poets use in practice. For example, the sentence “I went to the store” would be transformed to the sentence “to the store

Algorithm 1: Sentence Rearrangement Algorithm

Parse the sentence to generate a parse tree T ;
$S \leftarrow$ Leaf of T corresponding to the rhyming word ;
while *S is not the root of T* **do**
  $P \leftarrow$ parent of S;
  if *S is not the rightmost child of P* **then**
    Make S the leftmost child of P by switching it with the current leftmost child of P;
    $S \leftarrow P$
  end
end

I went" using the algorithm outlined above. This is a reasonable heuristic that preserves meaning, for the most part, while allowing for positions of words within the sentence to be re-arranged.

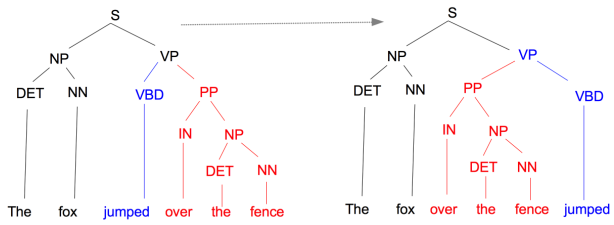

Figure 2. Illustration of Sentence Re-arrangment Algorithm

3.4. Score Candidates Sentences

In this step we evaluate the sentence pairs produced in step 3 and try to select the most promising candidates. Our first and foremost concern is producing syntactically correct sentences, as the heuristic we use for sentence rearrangement is far from perfect. Therefore we prefer candidate sentence pairs that required no sentence rearrangement, i.e. candidate sentences in which the last words, or their synonyms rhymed. If such an option is not available, we prefer candidates in which only one of the sentences has been rearranged. Lastly, we consider candidate pairs in which both sentences have been rearranged. Within each category we score each candidate by feeding both sentences through a trigram language model, taken from an old CS224N assignment source code[3], which we trained over a text corpus containing Wikipedia text, which is similar in syntax and vocabulary to the sentences in our test set. After recording the probability generated by the language model for each sentence we multiply the probabilities to produce the a cummulative score to both sentences. The language model is a great tool to eliminate syntactically weird sentences, as well as for for preferring commonly used words overly less words that are rarely used in day to day speech. We selected the 5 candidate sentence pairs with the highest score and present them to the user.

4. Experimental Setup

In order to evaluate our system we have created the following scale from 0-4, which we refer to as the cRhyme rate to evaluate the generated sentence pair for a given input sentence pair. A 0 on the cRhyme rate corresponds to the system not generating any matches, indicating that the synonym and rhyming dictionaries could not produce rhyming word pairs. A 1 on the cRhyme rate corresponds to candidate sentences that are syntactically incorrect. A sentence that has a score of 2 on the cRhyme rate is a sentence that is syntactically correct, if only barely so, but the meaning of the sentence has been altered significantly such that it does not make any sense. A cRhyme rate score of 3 indicates that the sentence is syntactically correct and preserves meaning, but in order to see this one needs to allow for some creative freedom. Finally, a cRyhme rate score of 4 indicates that the meaning of the sentence has been perfectly preserved, in addition to being syntactically correct.

cRhyme Rate	Interpretation
0	no candidates found
1	syntactically incorrect
2	syntactically correct, meaning is altered
3	syntactically correct, meaning preserved allowing for creative interpretation
4	syntactically correct, meaning preserved

Using this scale, we proceeded to obtain training and testing data for our system. In order to create a corpus of data we employed workers from Amazons Mechanical Turk service to input pairs of sentences that rhyme, and then alter one of the sentences such that the original pairs does not rhyme any more. Specifically, we asked the workers to paraphrase one of the sentences in the original pair in such a way that the precise word that made it rhyme with the other sentence is not present anymore. This required rearranging words, paraphrasing, and/or substituting synonyms. This yielded a set of sentence pairs that do not rhyme, and it is not trivial to discover a way to make them rhyme. However, using this setup, we were also guaranteed that a solution does exist, and is not too far out of reach for the system to tackle. We then fed this pair of sentences into our system and asked Amazon Mechanical Turks to manually score the output based on the cRhyme rate. We asked three workers to score every example and we used a majority vote scheme to decide on a score (or the highest score if no majority was achieved). We also keep around the original two sentences, our "gold standard" to compare the output with.

While we would have liked to tweak our system, and try different configurations and different features, due to the time and cost associated with acquiring labeled results, we decided to tweak the system ourselves and present the finished product to our judges to score.

5. Results and Analysis

Our results on 211 pairs of sentences in the test set were as follows:

Score	0	1	2	3	4
Count	29	24	26	77	55

Hence our average score on the test set was 2.52. In order to analyze this results we look at some of the sentences in the test set and how our system performed on them:

Input
the darkness comes with every night one sleep more we'll lose that battle
Output
the darkness comes with every night one sleep more we'll lose that fight

Judge Scores: 4, 4, 4. Final Score: **4**.
As we can see our system does well when all it needs to do is perform a synonym substitution transformation with no reordering. In the next two examples we see that our system does well with reordering and synonym substitution in some cases:

Input
rain rain go away come again another time
Output
go away rain shower come again another hour

Judge Scores: 4,4,3. Final Score: textbf4.

Input
well my daddy left home when i was three he didn't leave much to me - and to my remaining parent
Output
well my daddy left home when i was three he didn't leave much to my remaining parent - and to me we'll lose that fight

Judge Scores: 4, 4, 4. Final Score: **4**.
However, due to limitations/incompleteness of our rhyming and synonym dictionaries we sometimes fail to find a synonym that rhymes (when there exists one) as in the following case:

Input
i have a pretty little kitten she always walks around the cookhouse
Output
i have kitten pretty little amp she always around the cookhouse tramp

Judge Scores: 1, 1, 2. Final Score: **1**.

In this case the dictionaries fail to find "kitchen" as a synonym to "cookhouse", and hence the output became bizarre and even syntactically incorrect. This was the case for most of our low score outputs and is the single cause that adversely affects performance the most.

Another addition we made to our system is the use of hyponyms (a word of more specific meaning than a general or superordinate term applicable to it) and hypernyms (a word with a broad meaning that more specific words fall under). These expand our list of potential substitutions and hence make it more likely to find a pair of words that rhyme.

Input
orange blossoms they smell so pretty yet not enough to make me clever
Output
orange blossoms they so pretty snuff yet not to make me clever enough

Judge Scores: 4, 3, 4. Final Score: **4**.

In this example snuff, a hyponym, is replaced for smell. Since hyponyms have more specific meaning they are more likely not to alter the meaning of the sentence. Hypernyms, on the other hand, have broader meaning so they have the risk of diluting sentence meaning or even syntactic consistency when using them as in the following example.

Input
live the life to its best make it filled with light
Output
life to its best the motivation make it filled with illumination

Judge Scores: 2, 2, 3. Final Score: **2**.

Finally we can see that the language model does help in selecting better constructed sentences instead of misconstructed ones.

Input
three days ago we walked through mountains high i want to lay down and expire
Output
top choice: three days ago we walked through mountains high want to lay down and expire i **bottom choice:** we walked through mountains high ago three day i want to expire and down lay

Judge Scores: 4, 4, 3. Final Score: **4**.

The disadvantage of a language model is that infrequent words, although they make great rhymes, get pushed down quite low. Another disadvantage is that a language model only cares about whether the sentence formed is a correct English sentence, but does not care if the meaning was preserved or not. Our system currently does not have any protection against meaning change or corruption and can easily transform sentences changing their meaning. For example the sentence "Mary kicked John" can be reordered by our system to "John kicked Mary" which is a perfectly valid English sentence yet changes the meaning. Another problematic situation is the case of words with multiple meanings: we are in trouble if we chose to replace it with a synonym for another sense of the word. To preserve meaning, we mainly rely on the fact that at most we make one-word substitutions on both sentences and reordering, which does not change much of the sentence content and hence is likely not to change the meaning. Another limitation of our system is that it cannot do sophisticated paraphrasing of sentences in order to expand the set of possible paraphrasings. Our system only allows one for one word substitutions and reorderings, but does not allow many to one or one to many substitutions. As can be seen in the following example:

Input
i love you more than you will ever know i could only let you go always
Output
i love you more than you will ever know i could only let you always go

Judge Scores: 1, 2, 2. Final Score: **2**.

The more appropriate paraphrasing of the second sentence would have been "i will never ever let you go" but our system cannot support this.

6. Conclusion and Future Work

As one can see our system performs fairly well in the task of transforming pairs of sentences to rhyme. However, as we saw in the previous section there are some limitations to our system. First, the biggest factor that affects performance negatively is the lack of a better synonym dictionary. Many of the mistakes made by our system came about because the synonym dictionary could not find a specific synonym to a word that was beforehand known to rhyme with a word in the other sentence. By improving the synonym dictionary we could greatly improve the performance of our system.

A related but less significant problem with our system is synonym substitution for words with multiple meanings. Since our system currently cannot infer which of all the meanings a given word has, it can choose a synonym with a different meaning and hence makes the sentence pair rhyme but changes the meaning. This could be addressed by developing a context sensitive synonym resolution system that takes into account the context in which a word occurs, infers the words meaning, and looks up synonyms for that specific meaning.

Another improvement that can be made to the system involves the capacity to do more complex paraphrasing. Our system currently only handles one for one word substitutions. By adding more complex paraphrasing techniques that could do many to one, and many to many word substitutions we could increase the power of our system since many more candidates for rhyming are available.

Finally, in order to avoid permuting sentences like Mary kicks Tom to "Tom kicks Mary" (which are allowed by our language model but change the meaning), we could write some rules that prevent switching the subject and object of a verb.

References

[1] I. Androutsopoulos and P. Malakasiotis. A survey of paraphrasing and textual entailment methods. *J. Artif. Int. Res.*, 38(1):135–187, May 2010.

[2] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[3] Issao. Stanford language model implementation.

[4] B. Langenberger. The rhyming dictionary 0.9.

[5] G. A. Miller. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38:39–41, 1995.

[6] B. Spell. Java api for wordnet searching (jaws).

[7] R. L. Weide. Cmu pronouncing dictionary. 1994.

[8] S. Zhao, X. Lan, T. Liu, and S. Li. Application-driven statistical paraphrase generation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 834–842,

Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[9] S. Zhao, H. Wang, T. Liu, and S. Li. Extracting paraphrase patterns from bilingual parallel corpora. *Nat. Lang. Eng.*, 15(4):503–526, Oct. 2009.