RankLyrics: A Ranking-based System for Generating Song Lyrics

Shuo Zhang

I. Abstract

In this work we describe RankLyrics, a ranking-based system for generating song lyrics. Using a corpus of 600,000 lines of Rock lyrics that we built, we show that a simple baseline of lyrics generated by a N-gram language model can be improved by a statistical ranking model to achieve reasonable results with minimum hand crafted rules (including song specific traits like rhyming). First we describe building a corpus of about 600,000 lines of Rock lyrics by linking last.fm, MusicBrainz, and crawling lyricsWikia. For candidate line generation, we use the trigram language model estimated with SRILM using the lyrics corpus with Kneser-Ney smoothing. Given a line of lyrics, in order to pick the best next line, we then train a machine learning system to rank candidates of lyrics lines given the previous line and the current candidates. We use three types of hand engineered features: (1) language model (perplexity/log likelihood); (2) coherence (semantic similarity with tailor trained doc2vec paragraph vectors, word2vec average on the lyrics corpus, and with pre-trained Google News word embeddings); (3) song-specific features, including number of words in the lines, and rhyming. Using a subset of our lyrics corpus, we adopt a distant supervision approach to minimize annotation work. The resulting system is able to produce reasonable lyrics lines with coherence and (lots of) rhyming, and a preliminary evaluation shows a 22% error rate in a human judgment task using a data set consists of both human written lyrics and lyrics generated by RankLyrics.

II. Overview

A. Introduction and previous work

Song lyrics generation is a interdisciplinary task combining natural language generation (NLG) and music information retrieval (MIR). Previous work in this space has looked at different varieties of this task, including autonomous generation (by default, no input from user) and interactive generation with human writers in an assisting role [6] (which encompasses dimensions of human input such as sentiment/emotion, or musical constraints like rhythm). The current work targets autonomous lyrics generation without human input, and seeks to leverage recent advances in word vector embeddings to iteratively select the most syntactically sound, semantically coherent, and musically interesting lyrics lines from a pool of candidates generated by a simple ngram language model. It also leverages the distant supervision technique to utilize a large amount of training data during machine learning.

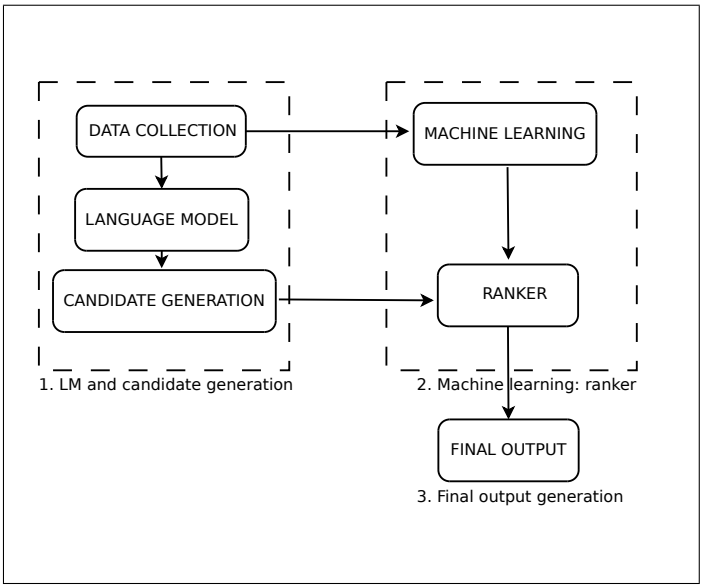

Fig. 1. System architecture

Although the system is trained on Rock, there is no inherent components that limit the system to generalize to other genres.

B. System architecture

There are four stages in building the RankLyrics system: (1) initial data collection of Rock lyrics corpus; (2) utilizing the corpus, training language modes and generate candidate lyrics lines; (3) using distant-supervision to perform machine learning to produce a ranker (probabilistic classifier) on a subset of the training corpus data; (4) finally using the ranker to rank lyrics candidates and generate the final output. Figure 1 shows the overall system architecture.

C. System dependency

The system depends on the following software and libraries: (1)kenlm (language modeling library); (2) SRILM (compiled binary is included); (3) Google News word embeddings vectors pre-trained by word2vec and downloadable from word2vec website; (4) Gensim (python NLP library); (5) Spacy (python NLP library). The system runs on Python 3.

III. Data collection

We rely on LyricsWikia as our source for Rock lyrics. Concretely, we utilized the Last.fm Dataset + LyricWikia + MusicBrainz train and test meta data sets (json dump) [1] as a

[1] See http://www.cs.ubbcluj.ro/ zbodo/lastfm.html

seed with linked metadata, and used artist and title information to query LyricsWikia, then parsed the HTML and scraped the lyrics texts. Using this method we collected lyrics from 15206 Rock songs in the training data set, and 1504 songs in the testing set. The total number of lines in the corpus is about 600,000.

The downloaded lyrics content includes some inconsistencies and required some preprocessing and cleaning. These include variations in character encodings, section headers (most songs do not indicate sections), non-alphanumeric symbols, etc. We inserted metadata information as a separator between each song. It is worth noting that if the corpus had contained more complete section headers, we could potentially train different models on verses vs. choruses or bridges, etc (assuming there are different attributes to them, which is not empirically attested yet). However, given the lack of these information, we simply removed all section headers as a preprocessing step.

IV. Language modeling and candidate generation

For lyrics candidate lines generation, we use a simple N-gram language model. There are several considerations behind this choice. First, previous works show N-gram models achieves reasonable results for the lyrics generation task [4]. Second, we consider the lyrics task to be a subtask of NLG, or an application of NLG in a particular domain. Within this domain, the task is less constrained by discourse coherence and semantic logic, which means a simple N-gram NLG component without complex content planning and CCG/CFG style template generation could suffice. Our observation of samples of real lyrics from the corpus shows that in this kind of poetic space, some creativity and incoherence is the text happen from time to time.

We consider two options for language modeling software: `SRILM` and `kenlm`. The latter is highly efficient although the data structure is not optimized for NLG. [2] We therefore use the SRILM for the current task. We experimented with several values of N and smoothing techniques (linear interpolation, good-turing, etc) to train N-gram LM, and we chose trigram LM with Kneser-Ney smoothing, which yields the lowest perplexity on the held-out test data set.

As a command line tool, SRILM is not easily interfaced with Python. Therefore we opt to use the `kenlm`'s Python interface to read and process the model produced by SRILM (which is ARPA format supported by `kenlm`).

V. Best next line: learning to rank candidates

A. Overview

We propose a machine learning task to solve the candidate lyrics line selection problem. For each candidate line and a given previous line of lyrics, can we train a classifier to predict if the current candidate line would be an optimal next line for the previous line? We then use the probability/confidence score outputted by the classifier to obtain a rank of all candidates, and we choose the most optimal candidate to be the next line.

[2]Comments by Kenneth Heafield.

This approach is conceptually somewhat similar to the approach taken in [2], who treated lyrics generation as a information retrieval problem where the first line is a query and the algorithm searches for the most relevant documents (next line) given to the query. However, [2] works at the level of sentence only (no word-by-word generation), has a different methodology (neural networks), and the current work is original in its design and is not attempting to replicating that work [3].

B. Feature engineering

There are three classes of features we extract from the pair input <$prev\_line$, $candidate_i$> for each i in [1,N] for the N candidates in the current iteration.

1) Language model feature: This feature captures how likely it is that the current sentence is a legal sentence given the training corpus of lyrics, by computing the log-likelihood of the current candidate line estimated according to the trigram LM we trained. The log likelihood is then normalized by sentence length.

2) Coherence features: Coherence features capture the fact that consecutive lines of lyrics are usually coherent in terms of their semantic and topical content. Here we use semantic similarity as a measure of coherence. Concretely, we use three kinds of sentence vector representations and compute semantic similarity (cosine distance) between the previous line and the current candidate: (1) we trained a `word2vec` model[3] on the lyrics corpus, and use the averaged w2v vector of all words as the representation for the whole sentence; (2) we trained a `doc2vec` model on the lyrics corpus, representing each line as a Paragraph Vector (sentence embedding). Paragraph Vector is a generalization of dense word embedding vectors to units beyond words (sentences, paragraphs, documents), and has been shown to perform well in tasks such as sentiment classification [1]; (3) similar to (1), except that instead of using a specifically trained model, we used pre-trained Google News Vectors as our word2vec model. Subsequent experiments showed that all three features contributes positively to improving the classification accuracy. Note that there are more discourse-oriented measures of coherence, such as Center Theory, Lexical Chains, and Vocabulary Introduction. We do not employ these due to time constraints, and also due to the fact that these features are more suitable for longer texts in discourses and other genres, but not in lyrics (our conjecture).

3) Song lyrics specific features: To capture features like rhyming, timing, number of words, etc., instead of crafting heuristics and rules, we also rely on machine learning to do the work and learn these features automatically. Therefore, we extract several song-lyrics-specific features from the pair of lyrics lines: (1) `is_rhyme` = 1 if the current candidate's final syllable rhymes with the previous line's final syllable, 0 otherwise; (2)`is_internal_rhyme` = 1 if the current candidate's final syllable rhymes with any one of the words in

[3]Given the time constraints on developing the current system, we mostly come up with ideas on what is the most sensible thing to do without much time to read up on literature.

TrainSize	TestAccuracy
4000	71.7
20000	72.8
30000	72.3
40000	72.8

TABLE I
CLASSIFIER ACCURACY

the current line, 0 otherwise; (3) log number of words of the candidate line; (4) log number of words in the previous line. To foreshadow the final results, the number 1 characteristics of the lyrics outputted from the system is that there are a lot of rhyming in the generated lyrics, greatly improving the text's quality as song-like. We attribute this success to the (apparent) abundance of rhyming phenomena in the training corpus.

C. Distant supervision

Instead of building hand annotations (which is also impossible and unnecessary), we leverage the natural structure of the lyrics corpus to serve as training data for our classifier. Specifically, we build positive and negative training examples using different methods within a distant supervision framework.

For positive examples, we sample all consecutive lines extracted from corpus (text is first segmented into chunks of consecutive passages, where each passage corresponds to a verse or a chorus passage without distinct labels), and extract features from each input pair to build the positive portion of the training set (N=100000).

The negative examples come from two sources using a technique similar to 'negative sampling'. First, within the lyrics corpus, we randomly sample two lines that are not adjacent to each other, and repeat this process for a specific number of time (N=50000); Second, we use SRILM to generate 50000 lines of lyrics from the trained trigram LM, then we ranked these by the log likelihood of the sentence assigned by the LM, and we take the bottom half and randomly sampled two lines from those 25000 lines (N=50000). These two sources ensure that the model is exposed to negative samples that are either not adjacent, or not adjacent AND bad (low log likelihood).

Together the training data set has 200000 training examples. Due to time constraints, we sampled 40000 (evenly balanced between classes) to use for training.

D. Training classifier

We train the classifiers using a SVM with RBF kernel. The SVM is configured to output probabilities to be used to generate a ranking. Classifiers are evaluated using a held-out data set of size 2000. When using 4000 training examples, the classifiers already reaches an accuracy of 71.7%. The final accuracy using all 40000 examples is 72.8%. Table I shows the results. As noted above, regarding the coherence features, even though the three features could be seen as somewhat redundant, when we used a subset of these features, the performance of the classifier consistently dropped.

Algorithm 1 lyrics generator architecture

```
1:  procedure GENLYRICS(numberOfLines,k=numberOfCandidates)
2:      firstLineCands ← genCandidates(k)
3:      firstLine ← maxLogLik(firstLineCands)
4:      prevLine ← firstLine
5:      counter ← 0
6:      repeat
7:          Cands ← genCandidates(k)
8:          nextLine ← selBestCand(prev)
9:          prevLine ← nextLine
10:         counter + +
11:     until counter = numberOfLines
```

VI. LYRICS GENERATION

A. Architecture

The lyrics generator architecture is illustrated in Algorithm 1. This algorithm illustrates the process of generating a passage of lyrics given the number of lines in the passage. Since we have no distinction between verses and choruses in our ranking model (they are trained on the same data), we simply cast a hard constraint on the number of lines in verses and choruses. For verse sections, we use a random number generator to generate the number of lines in the verse in the range of [5,11). For choruses, we have used a fixed number of 5 lines. Another hard constraint is that verses cannot have repeating lines whereas choruses may have repeating lines.

Without any use of user supplied seed, the first line of the lyrics is generated by selecting the best candidate with the highest log likeliehood assigned by the LM. Then, for each subsequent lines, we iterate through all candidates and use the previous line and current candidate as input to the ranking model. The model then produces a ranking and select the candidate with the highest probability. The process repeats until we reach the specified number of lines.

B. System efficiency

The performance of this architecture depends on the efficiency and quality of the LM text generation. The system has a hyperparameter of k, namely, number of candidates it generates for each line. In general, we have found the value between 100 and 200 to be sufficient and we use 150 as our final system value. Meanwhile, we have found that using SRILM, the sentence generation tends to be slow with these values of k. In particular, the inefficiency of the candidates generation process is the main component that slows down the system drastically. Therefore we have experimented with an alternative strategy: we use the pre-generated 50000 sentences as a pool for candidates, selected the top 23000 ranked by log likelihood, and we randomly sampled candidates from this pool. This results in great improvements in the speed of the system (8x faster). However, we observe that the resulting lyrics from this Fast system is not as interesting as the Slow system. Concretely, lyrics from the Fast system tend to be short and lack variation. Overall there is a tradeoff between speed and interestingness of the lyrics generated.

Another factor that slows the system down is the overhead of loading the binary Google News vectors (3.6G), which takes about a minute in a Intel core i5 laptop with 16GB RAM. To reduce this overhead, we tried using the .txt version of the Google News vectors (about 80MB). [4] However, after training the classifier on this model, the classification accuracy dropped to 70% (using 60000 training samples) and the resulted in obvious drop in the quality of generated lyrics. We attribute this to potentially the loss of information of the transformation (3.6G to 80M). Therefore currently we are still using the binary model.

VII. Evaluation

Subjectively, the final product of lyrics improves the baseline (=generated from LM directly) greatly. The sentences are mostly grammatical, have variations in length and syntactic construction, and captures many song-like qualities of the lyrics. The most successful traits of the product is the frequent use of rhyming, as noted above, greatly improving the lyrics quality. The semantic coherence level is also reasonable (although often not great) and much better than random, abrupt shift of topics. We attribute this improved level of coherence to semantic similarity (sometimes involving similar or identical words between lines). Meanwhile, we do note that there are several shortcomings of the lyrics generated, including preference toward short sentences (despite log likelihood normalized to sentence length) and sometimes lack of complex sentences. These shortcomings could be improved if the LM is able to generate more high quality sentences with greater probability.

The evaluation of automatically generated song lyrics is a challenge in itself. The objective measures used in previous work cover some aspects of the song traits, such as number of rhyming words. In the current work we have carried out preliminary subjective evaluation by mixing 13 generated lyrics with 6 real lyrics from the corpus and ask subjects to judge which ones are written by a human. The preliminary results show that the average error rate of judgment (N=6) is around 22%, with more false negatives than false positives, as predicted. This shows that while some machine generated lyrics is mistaken for human generated ones, there is also ambiguity in human written lyrics that led to the judgment that they are machine generated. We speculate that this is due to different characteristics of lyrics that distinguishes itself from other common natural language genres. Overall, this kind of subjective evaluation has many considerations that are yet to be taken in their designs in order to extract meaningful results from it. We leave that for future work.

VIII. Discussion and future work

In this work we demonstrated how a machine learning based ranking system can be used to leverage the power of NLP to select best candidates from sentences generated by a simple trigram language model to achieve reasonable performance in song lyrics generation.

[4]See https://github.com/zangsir/compdisc/blob/master/vectors/ for the data and the Python data structure that reads the data.

We discuss several points going forward for future works. First, we may experiment with more sophisticated NLG techniques to improve the quality of the sentences, including CCG/CFG frameworks. Second, we can also fine tune our current n-gram based LM to achieve better grammaticality. Third, there are also ways to improve the semantic coherence such as using topic models and to sample from clusters of topics. Fourth, we may include human-computer interaction functionalities (for instance, if a seed word or song is provided, then we will need to use something similar to TF-IDF/ICF to select most salient words to generate our lyrics). Fifth, we could add features to improve our ranking system, potentially in the realm of discourse coherence. Sixth, we could distinguish verses, choruses and bridges in our training (although it is to be empirically decided if that is beneficial).

Finally we also want to experiment with a completely different technique to approach this problem, namely, the so called seq2seq models[5]. These models represent the most recent advances in deep learning and its application in NLP, including state-of-the-art results in machine translation and chatbots. Seq2seq (sequence to sequence) models uses deep neural networks to encode and decode information in sequence data. We can envision a similar structure of problem to chatbots for song lyrics generation, where we have a trigger word and the model will yield a sequence (line of lyrics). Seq2seq models can be trained/implemented using frameworks such as `OpenNMT` and `Tensorflow`, etc.

References

[1] Le, Q., and Mikolov, T. Distributed Representations of Sentences and Documents. *International Conference on Machine Learning - ICML 2014 32* (2014), 1188–1196.

[2] Malmi, E., Takala, P., Toivonen, H., Raiko, T., and Gionis, A. DopeLearning: A Computational Approach to Rap Lyrics Generation. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16* (2016), 195–204.

[3] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Distributed Representations of Words and Phrases and their Compositionality. *Nips* (2013), 1–9.

[4] Nguyen, H., and Sa, B. Rap Lyric Generator. *Term paper from Stanford University* (2009), 1–9.

[5] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 2014), NIPS'14, MIT Press, pp. 3104–3112.

[6] Watanabe, K., Matsubayashi, Y., Inui, K., Nakano, T., Fukayama, S., and Goto, M. LyriSys : An Interactive Support System for Writing Lyrics Based on Topic Transition. 559–563.