Automatic Generation of Tamil Lyrics for Melodies

Ananth Ramakrishnan A
AU-KBC Research Centre
MIT Campus, Anna University
Chennai, India
ananthrk@au-kbc.org

Sankar Kuppan
AU-KBC Research Centre
MIT Campus, Anna University
Chennai, India
sankar@au-kbc.org

Sobha Lalitha Devi
AU-KBC Research Centre
MIT Campus, Anna University
Chennai, India
sobha@au-kbc.org

Abstract

This paper presents our on-going work to automatically generate lyrics for a given melody, for phonetic languages such as Tamil. We approach the task of identifying the required syllable pattern for the lyric as a sequence labeling problem and hence use the popular CRF++ toolkit for learning. A corpus comprising of 10 melodies was used to train the system to understand the syllable patterns. The trained model is then used to guess the syllabic pattern for a new melody to produce an optimal sequence of syllables. This sequence is presented to the Sentence Generation module which uses the Dijkstra's shortest path algorithm to come up with a meaningful phrase matching the syllabic pattern.

1 Introduction

In an attempt to define poetry (Manurung, 2004), provides three properties for a natural language artifact to be considered a poetic work, viz., Meaningfulness (M), Grammaticality (G) and Poeticness (P). A complete poetry generation system must generate texts that adhere to all the three properties. In this work, our attempt would be to generate meaningful lyrics that match the melody and the poetic aspects of the lyric will be tackled in future works.

According to on-line resources such as *How to write lyrics* (Demeter, 2001), the generated lyric must have Rhythm, Rhyme and Repetition.

One of the recent attempts for automatically generating lyrics for a given melody is the Tra-la-Lyrics system (Oliveira et al., 2007). This system uses the *ABC* notation (Gonzato, 2003) for representing melody and the corresponding suite of tools for analyzing the melodies. The key aspect of the system is its attempt to detect the strong beats present in the given melody and associating words with stressed syllables in the corresponding positions. It also evaluates three lyric generation strategies (Oliveira et al., 2007) – random words+rhymes, sentence templates+rhymes and grammar+rhymes. Of these strategies, the sentence templates+rhymes approach attempts for syntactical coherence and the grammar+rhymes approach uses a grammar to derive Portuguese sentence templates. From the demo runs presented, we see that the system can generate grammatical sentences (when using an appropriate strategy). However, there is no attempt to bring Meaningfulness in the lyrics.

2 Lyric Generation for Tamil

Tamil, our target language for generating lyrics, is a phonetic language. There is a one-to-one relation between the grapheme and phoneme. We make use of this property in coming up with a generic representation for all words in the language. This representation, based on the phonemic syllables, consists

Proceedings of the NAACL HLT Workshop on Computational Approaches to Linguistic Creativity, pages 40–46,
Boulder, Colorado, June 2009.

of the following three labels: *Kuril* (short vowel, represented by K), *Nedil* (long vowel, represented by N) and *Mei* (consonants, represented by M). For example, the word *thA-ma-rai* (*lotus*) will be represented as N-K-N (long vowel followed by short vowel followed by another long vowel). This representation scheme, herein after referred as *KNM* representation, is used throughout our system - training, melody analysis and as input to the sentence generation module.

3 Approach

Our approach to generating lyrics for the given melody is a two-step process (Figure 1). The first step is to analyze the input melody and output a series of syllable patterns in *KNM* representation scheme along with tentative word and sentence boundary. The subsequent step involves filling the syllable pattern with words from the corpus that match the given syllable pattern and any rhyme requirements. We approach the first aspect as a Sequence Labeling problem and use the popular CRF++ toolkit (Kudo, 2005) to label the input melody in *ABC* notation (Gonzato, 2003) with appropriate syllable categories (*Kuril*, *Nedil* and *Mei*). This system is trained with sample film songs and their corresponding lyrics (in *KNM* scheme) as input. The trained model is then used to label the given input melody. The syllable pattern, thus generated for the input melody, is provided to a Sentence Generation Module that finds suitable lyrics satisfying the following constraints: a.) Words should match the syllable pattern b.)The sequence of words should have a meaning. We achieve this by using the popular Dijkstra's Shortest Path Algorithm (Cormen et al., 1990) against a pre-built corpus of Unigram and Bigram of Words.

4 Melody Analysis

The goal of the Melody Analysis is to analyze the input melody and suggest a possible *KNM* representation scheme that will match the melody. Since our representation of melody is based on the *ABC* Notation (Gonzato, 2003), which is textual, we approach this problem as labeling the *ABC* notation using the *KNM* representation scheme.

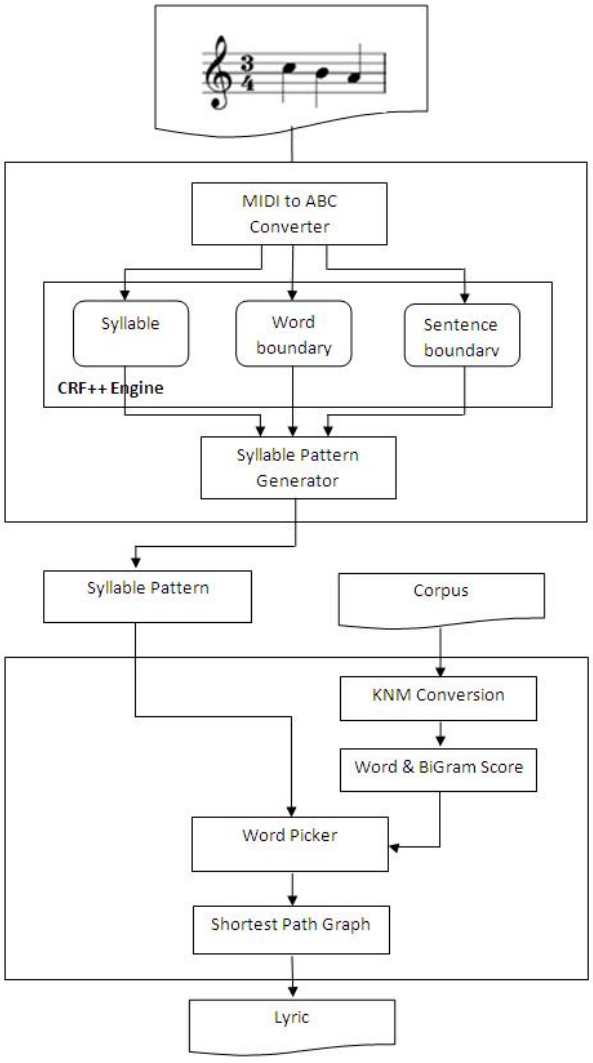

Figure 1. System Approach

4.1 Characteristics of Melody

Every melody follows a *Meter*, which provides the basic design principles in music. Some of the most frequently used meters we encountered in the film songs that we used are 2/4, 3/4, 4/4, 6/8, 9/8 and 12/8 – that indicate the number of Notes played in

the given interval. Each Note is represented by the character set A, B, C, D, E, F and G – which are called as main notes and A#, C#, D#, F# and G# - which are called Sharp Notes. Thus, for any given Meter in the melody, we can find the sequence of Notes with the corresponding duration for which the Note is played in that meter.

For the purpose of generating lyrics, we need to fit one syllable for each of the notes in the melody.

4.2 Conditional Random Fields

Conditional Random Fields(*CRF*) (Lafferty et al., 2001) is a Machine Learning technique that has performed well for sequence labeling problems such as POS tagging, Chunking and Named Entity Recognition. It overcomes the difficulties faced in other techniques like Hidden Markov Models(*HMM*) and Maximum Entropy Markov Model(*MEMM*).

(Lafferty et al., 2001) define Conditional Random Fields as follows: "Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph such that $\mathbf{Y} = (\mathbf{Y}_v)_{v \in V}$, so that $\mathbf{Y}$ is indexed by the vertices of $\mathbf{G}$. Then $(\mathbf{X},\mathbf{Y})$ is a conditional random field in case, when conditioned on $\mathbf{X}$, the random variables $\mathbf{Y}_v$ obey the Markov property with respect to the graph: $p(\mathbf{Y}_v|\mathbf{X},\mathbf{Y}_w,w\neq v) = p(\mathbf{Y}_v|\mathbf{X},\mathbf{Y}_w,w\sim v)$, where $w\sim v$ means that w and v are neighbors in $\mathbf{G}$". Here $\mathbf{X}$ denotes a sentence and $\mathbf{Y}$ denotes the label sequence. The label sequence y which maximizes the likelihood probability $p_\Theta(\mathbf{y}|\mathbf{x})$ will be considered as the correct sequence, while testing for new sentence x with CRF model . The likelihood probability $p_\Theta(\mathbf{y}|\mathbf{x})$ is expressed as follows.

$$p_\theta(\mathbf{y} \mid \mathbf{x}) \propto \exp\left(\sum_{e \in E,k} \lambda_k f_k(e, \mathbf{y}|_e, \mathbf{x}) + \sum_{v \in V,k} \mu_k g_k(v, \mathbf{y}|_v, \mathbf{x})\right)$$

where λ_k and μ_k are parameters from *CRF* model $\boldsymbol{\theta}$ and f_k and g_k are the binary feature functions that we need to give for training the *CRF* model. This is where we integrate the specific features of the problem into the machine learning models like *CRF*.

4.3 Feature Templates

There are three models that need to be learnt, viz, labeling notes with *KNM* scheme, identifying word boundaries and identifying line boundaries. We present below the features used to learn each of the above.

4.3.1 Learning *KNM* labels

In addition to the labels K, N and M, there are also other non-syllable features that need to be identified in the melody. Thus, the complete list of labels include, K, N, KM, NM, TIE, OPEN, CLOSE, PRE and BAR.

K – short vowel
N – long vowel
KM – short vowel followed by consonant
NM – long vowel followed by consonants
TIE – presence of a Tie in the meter
OPEN – opening of a tie
CLOSE – closing of a tie
PRE – Note that follows a tie
BAR – End of meter.

The following are the list of features considered:

- Current Note
- Previous Note + Current Note + Next Note
- Previous-to-previous Note + Previous Note + Current Note + Next Note + Next-to-next Note
- Current Note/Duration
- Previous Note/Duration + Current Note/Duration + Next Note/Duration
- Previous-to-previous Note/Duration + Previous Note/Duration + Current Note/Duration + Next Note/Duration + Next-to-next Note/Duration.

4.3.2 Word Boundary

Another important aspect in analyzing the melody is to spot potential word boundaries. While in many cases, the presence of bars could indicate potential word boundaries, there are also cases where a given word can span a bar (especially due to the presence of Ties). Hence, we need to explicitly train our system to identify potential word boundaries. The features used to identify the boundaries of words are mostly the same as for learning the *KNM* labels, but with the addition of considering two more previous notes along with their durations.

4.3.3 Sentence Boundary

As with Word Boundary ,we cannot assume sentence boundaries based on the musical notation and hence we also train our system to identify potential sentence boundaries. Sentence boundary identification happens after the word boundaries are identified and hence this additional feature is used along with the above-mentioned features for sentence boundary training.

5 Sentence Generation

The goal of the Sentence Generation module is to generate a meaningful phrase that matches the input pattern given in *KNM* scheme. For example, given an input pattern such as *'KMKM NKM NKN'*, it should generate a phrase consisting of three words each of them matching their respective pattern.

5.1 Corpus selection and pre-processing

Since we are interested in generating lyrics for melody, the corpus we chose consisted mainly of poems and short stories. The only pre-processing involved was to remove any special characters (such as "( ), $ % &, etc.) from the text. From this corpus, we index all Unigram and Bigram of Words. Each word is marked with its *KNM* syllable pattern and their frequency of occurrence in the corpus. The Bigram list contains only the frequency of occurrence.

5.2 Graph Construction

Given an input pattern (say *'KMKM NKM NKN'*), we construct a directed graph with the list of words satisfying each pattern, as represented by Figure 2.

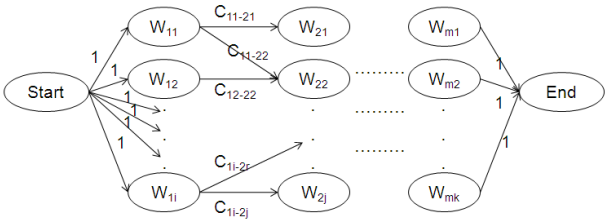

Figure 2. Graph Construction

The edge from word W_{ij} (of, say pattern *KMKM*) to W_{rs} (of, say pattern *NKM*) is weighted based on the frequency values collected from the corpus and is calculated as follows:

$$P(W_{rs} / W_{ij}) = \frac{\#\ (W_{ij} \text{ followed by } W_{rs})}{\#\ (W_{ij})} \quad \textit{(Eqn. 1)}$$

Since the Shortest Path Algorithm picks the path with the least cost, we need to weight the edges in such a way that a higher probability sequence gets the least cost (C). Thus, we measure $Cost(W_{rs}/W_{ij})$ as:

$$Cost(W_{rs}/W_{ij}) = 1 - P(W_{rs}/W_{ij}) \quad \textit{(from Eqn. 1)} \quad \textit{(Eqn. 2)}$$

By default, the cost from the START node to the first list of words and the cost from the last list of words to END node is fixed as 1.

5.3 Preferential selection of paths

One of the shortcomings of using the Shortest Path Algorithm is that, for the given input pattern and the given Corpus, the algorithm will always generate the same phrase (with the least cost). In addition to

this problem, when the melody demands, we need to generate rhyming words. Lastly, we need to handle the case where the corpus may not have a phrase that matches the complete pattern. We tackle all the above issues by biasing the Shortest Path Algorithm by changing the cost of the edges.

5.3.1 Bias initial word

In order to generate different phrases for the same pattern (say *KMKM NKM NKN*), we pick a random word that matches the initial pattern (*KMKM*) and fix the cost of the edge from START to the random word to 0. As the default cost from START to all leading words is 1, this biases the algorithm to find a pattern that starts with the random word. However, if there exists a phrase, whose "overall cost" is still less than the one starting with the random phrase, the algorithm will output the same phrase. In order to avoid this, we provide multiple random words and pick the one that truly generates a unique phrase.

5.3.2 Rhyming Words

When there is a need to generate phrases that rhyme with any previously generated phrases, especially in line endings, we use the same biasing technique to prefer certain words over others. The motivation to concentrate on line endings is based on our assumption that the notes in melody would be similar for the rhyming words and thus our representation scheme involving Mei(M) (consonants) would handle the stressed syllables. The path finding algorithm, can take as input a word and a position, with which the new phrase should rhyme in the given position. In this case, we generate all the words that rhyme with the given word by using the *Maximum substring matching technique*. That is, the word with the maximum substring common to the input word, in word endings, is considered as a rhyming word. For example, given an input word '*kOyil*' (*temple*), the rhyming words would be '*vAyil*' (*gate*) and '*veyil*' (*sun*). As can be seen, both the words have the suffix '*yil*' common with the input word. Thus, as earlier, the cost of the edges in the paths leading to such rhyming words will be set to 0, thus biasing the algorithm to pick these paths. One another way would be have only those nodes corresponding to the rhyming words (discarding other non-rhyming words). However, in the case where no rhyming words are present in the corpus, this approach can lead to a graph with an incomplete path. Hence we use the approach of biasing the graph paths that can pick the rhyming words, if present and provide a non-rhyming word, if none was available.

5.3.3 Edit-Distance Matching

There can also be cases when there is no phrase that exactly matches the given input pattern sequence, though the corpus might contain individual words matching each pattern in the sequence. In this case, we relax the matches using the *Edit-Distance* metric. Thus, for the given pattern *NKN*, we also list words that match *NKK*, *KKN*, etc. Since the input patterns are deemed to fit the given melody, an Edit-Distance Matching can turn up words that need not match the given melody and hence should be used only when there are no phrases matching the input pattern. Another approach, though practically not possible, is to have a "*big enough corpus*" that contains at least one phrase matching each pattern.

6 Experiments

We conducted the experiments as two separate steps, one for the *CRF* engine and another for the Sentence Generation module.

For the *CRF* engine, we collected and used Tamil film songs' tune and lyrics, as they were easily available from the web. The tunes were converted to the *ABC* notation and their lyrics were converted to the *KNM* representation scheme. The notes from the tune and the syllables in the corresponding lyric

(in their respective representation schemes) were manually mapped with each other. An example training file for the *CRF* engine for learning the *KNM* representation scheme is presented below:

Note	Duration	Label
B	½	K
C	½	N
B	½	K
A	½	KM
G	½	K
-	0	tie
[	0	open
A	½	pre
G	½	K
]	0	close
B	4	K

Table 1. *KNM* scheme learning – training file

Similarly, for the word boundary identification, the same input is used but with the labels corresponding to word boundaries such as W-B (word beginning), W-I (word intermediate), etc. (Table 2):

Note	Duration	Label
B	½	W-B
C	½	W-I
B	½	W-I
A	½	W-I
G	½	W-B
-	0	Tie
[	0	open
A	½	pre
G	½	W-I
]	0	close
B	4	W-I

Table 2. Word boundary learning – training file

For sentence boundary identification, the output from the word boundary identification is used and hence it is run after the word boundary identification is complete. Thus, the input to the CRF engine in this case would be like the one in (Table 3), with labels corresponding to sentence boundary such as S-B (sentence beginning) and S-I (sentence intermediate):

Note	Duration	Word Boundary	Sentence Boundary
B	½	W-B	S-B
c	½	W-I	S-I
B	½	W-I	S-I
A	½	W-I	S-I
G	½	W-B	S-I
-	0	Tie	S-I
[	0	Open	S-I
A	½	Pre	S-I
G	½	W-I	S-I
]	0	close	S-I
B	4	W-I	S-B

Table 3. Sentence boundary learning – training file

For the Sentence Generation module, we used short stories, poems and Tamil lyrics across various themes such as love, appreciation of nature, patriotism, etc. From this, all the special characters were removed and the list of Unigram and Bigram Words were collected along with their frequencies.

Based on the limited experiments performed on the trained CRF model, we observe that the feature set presented for Syllable identification seem to perform reasonably and identifies the syllables with 70% accuracy for manually tagged melodies. However, we could not objectively evaluate the Word and Sentence Boundary identification process as the resulting boundaries can also be considered as valid boundaries. In general, the word and sentence boundaries are the choice of the lyricist and hence the results can be considered as another valid way to generate lyric. Also, we feel that the number of training samples (10 melodies) supplied for training

the CRF engine is very less for it to reasonably learn the nuances that are present in real-word lyrics.

Some of the syllable patterns identified from the tune and the corresponding sentences generated are given below:

Pattern: '*KK KK KKK*
NKKM KMKK'
Output: ஒரு சிறு வயது
ஞாபகம் வந்தது
Translation: *In small age*
I recollected

As the syllable patterns get longer, we had to resort to using Edit Distance in order to find matching sentences. One such output is presented below:

Pattern: '*KMKMKM KMKM NKN*
NKMKM NMKKM NKN'
Output: தமிழில் இங்கு காணலாம்
என்று முறைப்புடன் சொல்லி
Translation: *We can see here in Tamil*
Proclaiming aloud

7 Limitations and Future Work

From the initial set of experiments, we see that it is possible to generate a syllable pattern that closely matches the input tune. Currently, we do not consider the identification of strong beats in the melody and are expecting the presence of Mei (M) to take care of stressed syllables. We also expect the same strategy to work for other South Indian languages as well. The current Lyric Generation algorithm is simplistic, in that it can generate short meaningful phrases, but generating longer phrases require adding constraints (such as closest matching patterns) that defeats the purpose of matching with the tune. Also, the current method generates phrases that are independent of the previous phrases. This leads to lyrics that are meaningful in parts, but meaningless on the whole.

Future work can involve introducing "*semantic similarity*" across phrases in a lyric, thereby generating lyrics that provide a coherent meaning. Also, experiments can be conducted with different domain corpus to generate lyrics for a given situation (such as Love, Death, Travel, etc.) Other sentence generation strategies, such as an *Evolutionary Algorithm* (as suggested in (Manurung, 2004)) can also be attempted. Once a coherent meaningful lyric is generated, further improvements can be towards incorporating poetic aspects in the lyric.

References

Demeter. 2001. *How to write Lyrics* http://everything2.com/index.pl?node=How to write lyrics.

Guido Gonzato. 2003. *The ABCPlus Project* http://abcplus.sourceforge.net.

Hanna M. Wallach. 2004. *Conditional Random Fields: An Introduction.* Technical Report MS-CIS-04-21. Department of Computer and Information Science, University of Pennsylvania.

Hisar Maruli Manurung. 2004. *An evolutionary approach to poetry generation.* Ph.D. Thesis, University of Edinburg.

Hugo R. Goncalo Oliveira, F. Amilcar Cardoso, and F. Camara Perreira. 2007. *Tra-La Lyrics: An approach to generate text based on rhythm.* Proceedings of the Fourth International Joint Workshop on Computational Creativity, IJWCC'07, London:47-54.

Hugo R. Goncalo Oliveira, F. Amilcar Cardoso, and F. Camara Perreira. 2007. *Exploring difference strategies for the automatic generation of song lyrics with Tra-La Lyrics.* Proceedings of the Portuguese Conference on Artificial Intelligence (EPIA 2007), Guimarães, Portugal:57-68

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence data.* Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Willamstown, MA, USA:282-289

Taku Kudo. 2005. *CRF++: Yet Another CRF toolkit.* http://crfpp.sourceforge.net.

Thomas H. Cormen., Charles E. Leiserson., Ronald L. Rivest. 1990. *Introduction to Algorithms.* Prentice-Hall: 527-531.