CSE3013

Artificial Intelligence

Final Report

Prof: Ilantheneral KPSK

Submitted by:
RAJVARDHAN DIXIT -16BCB0109

Project Title:

Clean Lyrics Generation using NLTK and LSTM

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Index

1. Abstract

It's a neural network that has been trained on Kanye Ist's (very famous American singer and music producer) discography, and can use any lyrics you feed it and write a new song *word by word* that rhymes and has a flow (to an extent). It is basically a markove chain will look at the lyrics you entered and generate new lines. Then, it feeds this to a recurrent neural net that will generate a sequence of tuples in the format of:

(Desired rhyme, desired count of syllables).

The program will then sift through the lines the markove chain generated, and match the lines to their corresponding tuples. The result is the rap.

2. Introduction

Neural Network is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. It learns by example.

The history of neural networks that was described above can be divided into several periods. There Ire some initial simulations using formal logic. McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks Ire based on simple neurons which Ire considered to be binary devices with fixed thresholds. The results of their model Ire simple logic functions such as "a or b" and "a and b". Significant progress has been made in the field of neural networks-enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neurally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

To design a neural network that has been trained on by songs of any artist by feeding lyrics and produce a new song word by word that rhymes and has a flow. This is a new way of applying machine learning attributes to a new region of people’s interest. Having a system that comes up with new songs which is similar to your liking boosts up a person’s mood. Having machines serve as composers actually opens the music world up to a “new creativity” that isn’t possible otherwise. Latest researches are going on to write and publish new albums which give tough competitions to old hits by learning the lyrics of those songs. One of them is carried out by Sony CSL known as Flow Machines.

3. Literature Review Summary Table

Paper	Methodology	Conclusion
Text based LSTM networks for Automatic Music Composition	• The proposed method includes two LSTM layers and the Keras deep learning framework. • The proposed network is designed to learn relation- ships within text documents that represent chord progressions and drum tracks in two case studies. • The proposed system can be used for fully automatic composition or as semi- automatic systems that help humans to compose music by controlling a diversity parameter of the model.	They introduced an algorithm of text-based LSTM networks for automatic composition and reported results for generating chord progressions and rock drum tracks. Word-RNNs shoId good results in both cases while char-RNNs only successfully learned chord progressions. The experiments show LSTM provides a way to learn the sequence of musical events even when the data is given as text.
Lyric Generation using artificial intelligence	• Lyric Generator is application that can be used to generate lyrics automatically i.e. all by itself. • Here the user only needs to describe the scenario of the lyric that is the mood, relationship, character description, rhyming, similarities, and inspirations for the lyrics. • These are certain criteria by which the computer or the application understands about what exactly I want it to write for us. • It is very essential that I provide lot of knowledge to the system prior to the creation of the application. • Updating the database of lyrics is also important. The database may vary from one language to the other but the concept remains the same.	In this paper, a creation technique has been proposed which can be used to create a lyric by using the ten prominent features of the lyrics. These Features have a pattern and I can create various patterns using appropriate algorithms. It is important to specify the meaning and usage of a word in a computer database. This application can be used to generate lyrics to any language provided the database has been loaded correctly. Future work includes techniques for finding similes and metaphors.

A survey on intelligent poetry Generation – Languages, Features, Techniques, Reutilization and Evaluation	• Poetry generation is becoming popular among researchers of Natural Language Generation, Computational Creativity and, broadly, Artificial Intelligence. • To produce text that may be regarded as poetry, computational systems are typically knowledge-intensive and deal with several levels of language. • This paper surveys intelligent poetry generators around a set of relevant axis – target language, form and con- tent features, applied techniques, reutilisation of material, and evaluation – and aims to organise work developed on this topic so far.	Intelligent poetry generators Ire surveyed in this paper, around a set of relevant axis where alternative approaches have been explored. Poetry has been automatically generated in different languages and forms, considering different sets of features, and through significantly different approaches. Poetry generators have been developed with different goals and intents, each with their stronger and laker points, which adds to the subjectivity involved in the evaluation of poetry, even for humans.
Algorithmic Song Writing with ALYSIA	• This paper introduces ALYSIA: Automated Lyrical Song- writing Application. • ALYSIA is based on a machine learning model using Random Forests, and I discuss its success at pitch and rhythm prediction. • Next, it is shown how ALYSIA was used to create original pop songs that Ire subsequently recorded and produced. • Finally, vision for the future of Automated Song writing for both co-creative and autonomous systems has been discussed.	Algorithmic song writing offers intriguing challenges as both an autonomous and a co-creative system. An autonomous song writing system producing works on par with those of expert human songwriters would mark a significant achievement.
Automatically Generating Rhythmic Verse with Neural Networks	• Two novel methodologies for the automatic generation of rhythmic poetry in a variety of forms have been proposed. • The first approach uses a neural language model trained on a phonetic encoding to learn an	First, they developed a neural language model trained on a phonetic transliteration of poetic form and content. Although example output looked promising, this model was limited by its inability to generalise to novel forms of verse. They then proposed a

	implicit representation of both the *form* and *content* of English poetry. • This model can effectively learn common poetic de-vices such as rhyme, rhythm and alliteration. • The second approach considers poetry generation as a constraint satisfaction problem where a generative neural language model is tasked with learning a representation of content, and a discriminative lighted finite state machine con- strains it on the basis of form.	more robust model trained on unformed poetic text, whose output form is constrained at sample time. This approach offers greater control over the style of the generated poetry than the earlier method, and facilitates themes and poetic devices.
Automated Composition of Lyrical Songs	• They address the challenging task of automatically com- posing lyrical songs with matching musical and lyrical features, and present the first prototype to accomplish the task. • The focus of this paper is especially on generation of art songs. • The proposed approach writes lyrics first and then composes music to match the lyrics. • Some example songs composed by M.U. Sicus have been proposed, first steps towards a general system combining both music composition and writing of lyrics have been outlined.	They have proposed the task of generating lyrical songs as a research topic of computational creativity. This topic has received only little attention in the past although both music composition and poetry/lyrics generation have been studied on their own. An automatic generation procedure of lyrical and musical content also offers interesting possibilities for medicalization of data
Automatic Generation of Poetry inspired by Twitter Trends	• This paper revisits PoeTryMe, a poetry generation platform, and presents it's most recent instantiation for producing poetry inspired by trends in the Twitter social network. • The presented system searches for tweets that mention a given topic,	A new instantiation of PoeTryMe, a poetry generation platform, was presented. The singular feature of the presented system is that its poetry is inspired by Twitter trends, more precisely, words that are associated with those.

	extracts the most frequent words in those tIets, and uses them as seeds for the generation of new poems. • Generation is performed by the classic PoeTryMe system, based on a semantic network and a grammar, with a previously used generate &test strategy. • Illustrative results are presented using different seed expansion settings.	
GhostWriter: Using an LSTM for Automatic Rap Lyric Generation	• This paper demonstrates the effectiveness of a Long Short-Term Memory language model in our initial efforts to generate un-constrained rap lyrics. • The goal of this model is to generate lyrics that are similar in style to that of a given rapper, but not identical to existing lyrics: this is the task of ghostwriting. • Unlike previous work, which defines explicit templates for lyric generation, this model defines its own rhyme scheme, line length, and verse length. • The experiments show that a Long Short-Term Memory language model produces better "ghost-written" lyrics than a baseline model.	The performance of the LSTM model has been compared to a much simpler system: an n-gram model. The results of our experiments show that, as an unsupervised, non-template model, the LSTM model is better able to produce novel lyrics that also reflect the rhyming style of the target artist. In future work, I plan to use more data to train our model, making it easier for our system to actually identify rhyming pairs and use them in new con- texts.
Automatic Generation of	• It is interesting to see if machine could learn Bob Dylan's poetic style by looking at his lyrics.	Fot N-he training perplexity decreases as N increases because larger N leads to feIr choices of the next word, hence higher probability to select the right word.

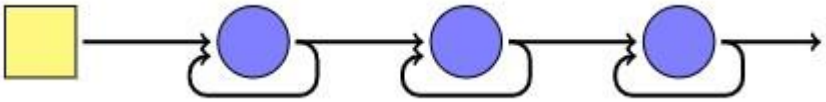
Lyrics in Bob Dylan's Style	• In this project, N-grams and Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) have been used to model Dylan's lyrics, and eventually use the algorithms to generate samples of lyrics in Bob Dylan's style.	Character-level RNN seems good at capturing the grammar of the sentences, but may be Iak in generating text that makes sense in the context.
SMUG: Scientific Music Generator	• Music is based on the real world. Composers use their day-to-day lives as inspiration to create rhythm and lyrics. • Procedural music generators are capable of creating good quality pieces, and while some already use the world as inspiration, there is still much to be explored in this. • A system has been described to generate lyrics and melodies from real-world data, in particular from academic papers. • Through this they want to create a playful experience and establish a novel way of generating content. • For melody generation, they present an approach to Markov chains evolution and briefly discuss the advantages and disadvantages of this approach.	They have presented a method for creating melody and lyrics using real-world data. To do so, they developed a musical generator that evolves Markov chains to create melodies, and a lyric generator, that extracts content from academic papers and transforms them into songs. Hence there is fully functional system that completes tasks, taking an academic paper in PDF format and outputting a melody and the according lyrics. The generator seems to produce interesting music/lyrics combinations, but they still have to conduct further studies to prove their interestingness.

4. Proposed Architecture and Framework

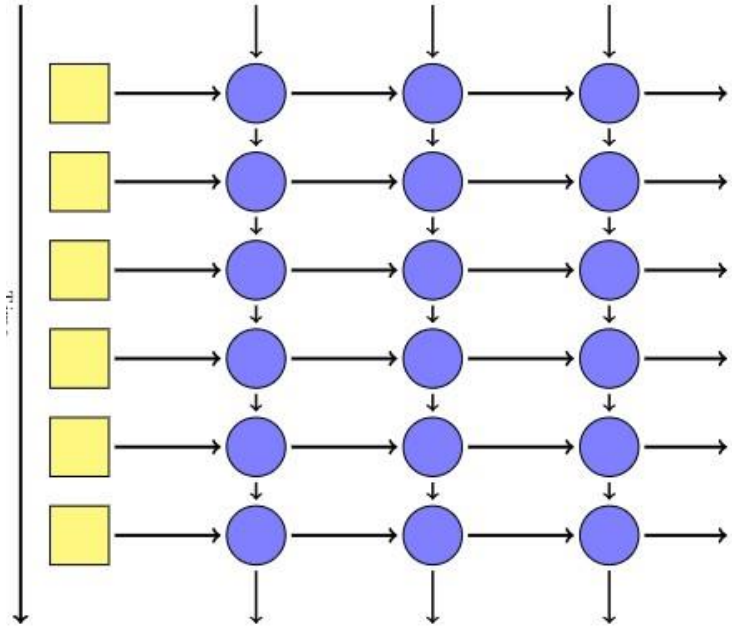
Recurrent Neural Networks

The basic feed forward network, there is a single direction in which the information flows: from input to output. But in a recurrent neural network, this direction constraint does not exist. There are a lot of possible networks that can be classified as recurrent, but I will focus on one of the simplest and most practical.

Basically, what I can do is take the output of each hidden layer, and feed it back to itself as an additional input. Each node of the hidden layer receives both the list of inputs from the previous layer and the list of outputs of the current layer in the last time step



This, after unwrapping along the time axis looks like this:



In this representation, each horizontal line of layers is the network running at a single time step. Each hidden layer receives both input from the previous layer and input from itself one time step in the past.

The power of this is that it enables the network to have a simple version of memory, with very minimal overhead. This opens up the possibility of variable length input and output: I can feed in inputs one-at-a-time, and let the network combine them using the state passed from each time step.

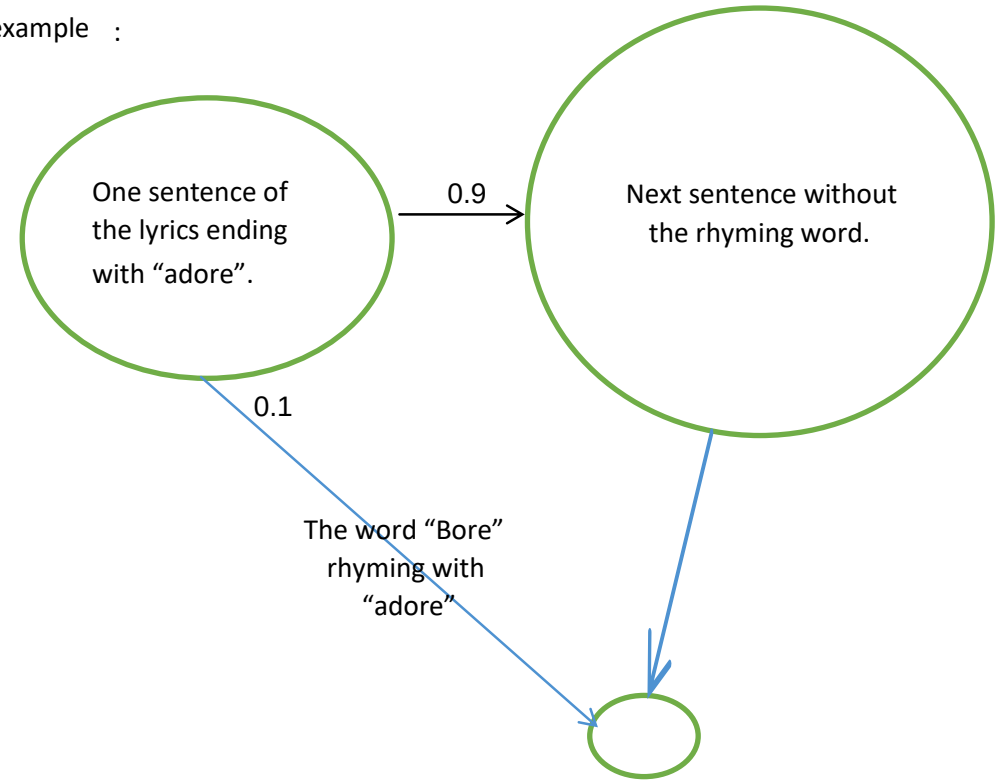
One problem with this is that the memory is very short-term. Any value that is output in one time step becomes input in the next, but unless that same value is output again, it is lost at the next tick. To solve this, I can use a Long Short-Term Memory (LSTM) node instead of a normal node. This introduces a “memory cell” value that is passed down for multiple time steps, and which can be added to or subtracted from at each tick.

About the process used in the project

First of all the dataset is cleaned using Natural Language Processing Tool Kit. I am using two datasets of positive words and negative words, which are compared with each line in the dataset. According to the calculated score of each word, negative sentences are removed from the dataset. This new filtered dataset is given to the neural network for computation. Stemmer function is used to stream the words . Tokenize function keeps the needed words and removes other things.

A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. Thus the working of the model through Markov chain is through the states of lyrics obtained from numerous songs collected. This being said, the probability of every word occurring in a specified pattern is calculated, the rhyming words are played after certain time with huge probability and the lyrics are composed.

For example :



Therefore, I saw that according to data collected from numerous lyrics of the genre “rap” of the same singer, there is a high probability of getting two rhyming words separated by sentences, rather than having both words said one after the other.

The probability of “Bore” being joined with “adore” = P (Adore->Bore) = 0.1. The probability of

“Bore” being joined with a sentence first = P (Adore->”some sentence” ->Bore) = 0.81.

Clearly the Markov chain helps to follow the specific rules of rap music, following the sequence of pattern regular rap music follows and the way a composer would write down the lyrics for it to rhyme.

5. Work to be done and implementation

This project is carried out in the below mentioned stages:

- **Data Preparation –** This stage involves collecting, compiling and cleaning the lyrics I want to feed into the neural networks.
- **Training –** Train the network with the fed lyrics.
- **Generating Lyrics - from** the already trained network.
- **Making Music –** Producing songs by using a text-to-speech conversion over a beat.

I am using Keras library. This is a deep learning and high level neural networks library in Python with various models.

Markovify library which uses Markov chains to generate random semi-plausible sentences based on an existing text.

pyttsx is a Python package supporting common text-to-speech engines

I am implementing the project in Python on Anaconda Navigator interface.

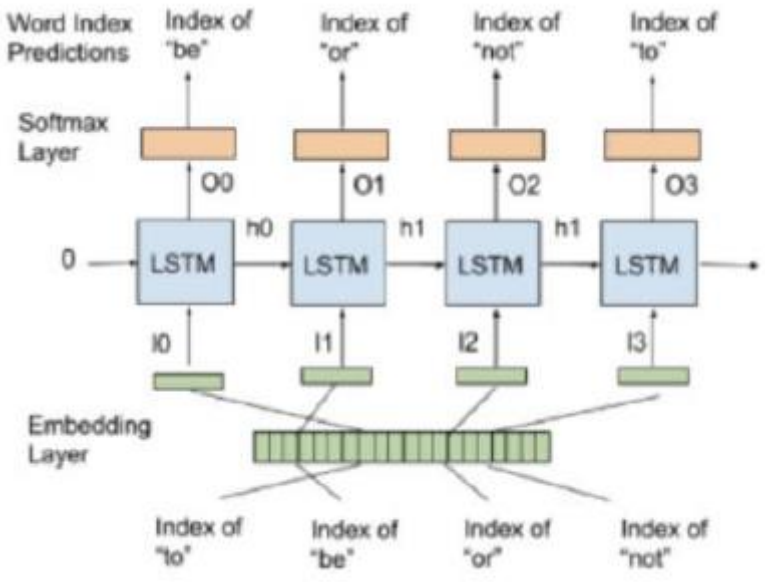
Code Description

With this project I begin with training a recurrent neural network on a very tough task of language modelling. I implement this on a set of lyrics by a rap artist, namely, Kanye west. I take his discography as a text file which becomes the input of the program.

The final aim is to fit a probabilistic model whose task would be to assign probabilities to the sentence and I use a markove chain model for that. Now this will be carried out by predicting the next words of the lyrics given the whole input of the discography of lyrics which will be used to learn i.e. use as a history of previous words.

The core of this model will be made of five LSTM cells. LSMT being the most widely used RNN and they are way better at capturing the dependencies, in this case from the set of lyrics inputted as the training set. These LSTM cells will invite words and then help predict the outcome of the next word by using the probabilities.

Now, by the method of design the outputs from the LSTM cells or in RNN in general are dependent on the set of random inputs. Although this makes back propagation difficult but in our case five epochs are done. One epoch will occur after one forward pass of the words followed by a step of back propagation to take the error into account and reduce it. That means, one epoch will occur after one forward and one backward pass.



After a sequence of tuples is generated this will be in the form of (desired rhyme, desired count of syllable). The program will then sift through the lines the markove chain generated, and match the lines to their corresponding tuples.

Implementation Explanation

I took a set of lyrics or a set of text as an input which I eventually feed to the Recursive Net.

The code is designed in a way that it takes the lyrics that I have entered and a Markov Chain looks at it and generates new lines.

Further, this will be fed to a Recursive Neural Network which will create tuples in the form of (desired rhyme, desired count of syllable)

For desired syllable

The neural net generates all the sets of syllables in the input set. To perform this task it takes the following steps:

1. It strips out all the punctuations from the input set to avoid any discrepancy.
2. It then takes the set of all the vowels and picks out words with different vowel-vowel or a vowel-consonant or a consonant-consonant combination in them.
3. It ignores those rhyming syllables which are not a combination of a pair-pair syllable rhyme. For example: It will not match the syllable sound of “e” with the syllable sound of “le” even though they have the same sound because they are not in accordance with a pair-pair syllable match scenario.

For desired rhyme

A rhyme scheme is the pattern of rhymes that is being used, whenever I am constructing a musical flow.

I need to understand how the genre of rap music is based on rhyming lyrics, be it words or syllables. The better the rhymes the better the lyrics.

So I basically desire a combination of rhymes in a sentence to be able to create a rap song. To do just that, I take the following steps:

1. I decide maximum syllables wanted in a single line. This is important as this will in turn determine how many rhyming syllables I will have in our lines.
2. It will then find out all the rhyming syllables in the set of generated syllables and identify them to be used in the, to be generated rap song.
3. The Recursive Neural Network will go through five Epochs to minimize the loss in each cycle and in turn find out the best combinations for the, to be generated rap song.
4. All these rhymes will be then saved in a .rhyme type file and be later used in the program.

To understand this better let us take a verse and try and match all the pair-pair syllable sounds in it; that is, a two pair syllable sound.

Penitentiary chances, the devil dances/ 12, —2/
and eventually ansIrs, to the call of Autumn/ -12,—–3/
all of them fallin', for the love of ballin'/ 33—-3/
got caught with 30 rocks the cop look like Alec Baldwin/ —4, 4–3/

We can see here that this piece of lyrics contain 2 consecutive multi-syllable rhymes in the beginning and then ending them with a rhyme that contains the same number of syllable as the previous of the same rhyme.

5. Limitations

1. **If the dataset is small, the machine will not be able to learn anything from it**, because the idea of a data set is to train the machine, so a large data set is needed. Even if the machine does learn anything, it will over fit or there will be a certain outlier condition and the answer won't be in accordance to our design.

2. **Our model's predictions are accurate, but often recycled.** It's important to note that many of our predicted lines turned out to be nearly identical to lines Logic has actually written, i.e. half of a line from one verse/song combined with half of a line from another verse/song. This is to be expected, as using bigrams yields less variability in predicted words due to basing predictions off the previous two words instead of the one most recent, resulting in sequences of three or more words coming from the same Logic lyric. To put it simply, using bigrams instead of single words increases readability and similarity to Logic's style, but decreases creativity.

3. **Our model is slower and generates less output.** The unigram model runs faster because the dictionary object representing its Markov Chain has far fewer keys. Our model has so many more keys because it has to process tuples of two words. Furthermore, as mentioned before,

there were times when I received very little to no output, and generally I received less than I did from the unigram implementation. This can be attributed to the smaller number of possibilities for the next word when I based it off the previous two words.

The range of algorithms available for ML problem solving is astounding. Random forests, support vector machines, neural networks, and Bayesian estimation methods – the list goes on (and on, and on). The question of what algorithm is best for a given ML problem, however, is often less impactful than I might think.

It's true that some approaches, on some questions, will work better than others. In some cases, this difference can even be quite distinct. However, in my experience it's been quite rare that one modelling approach will strictly dominate all other options in answering a given ML question. Therefore I also have faced cons of applying the markove model here.

6. References:

1. ://web.stanford.edu/class/cs221/2018/restricted/posters/ruoxi17/poster.pdf

2. cs229.stanford.edu/.../LiLiangLiuAutomaticGenerationofLyricsinBobDylan Style- rep...

3. Jack Hopkins et al. ―Automatically Generating Rhythmic Verse with Neural Networks‖

4. Hugo Oliveira ―A survey on intelligent poetry Generation – Languages, Features, Techniques, Reutilization and Evaluation‖, in International Natural Language Generation Conference, September 2017

5. Peter et al."GhostWriter: Using an LSTM for Automatic Rap Lyric Generation,‖ in Conference on Emperical Methods in Natural Language Processing, September 2015.

6. Dongzhou et al. ―Automatic Generation of Lyrics in Bob Dylan's Style.

7. Keunwoo et al. ―Text Based LSTM networks for Automatic Music Composition‖, in arXiv, April 2016.

CODE :

Sentence_Clasifier.py

```
import nltk
import string
pre=''
sum1=0
pos=[]
#from win32com.client import Dispatch
counter = 1
croppedResponse = ' '
#speak = Dispatch("SAPI.SpVoice")
path='chat.txt'
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()
from sentence_corps_1 import training_data
corpus_words = {}
class_words = {}
classes = list(set([a['class'] for a in training_data]))
for c in classes:
    class_words[c] = []
for data in training_data:
    for word in nltk.word_tokenize(data['sentence']):
        if word not in ["?", "'s"]:
            stemmed_word = stemmer.stem(word.lower())
            if stemmed_word not in corpus_words:
                corpus_words[stemmed_word] = 1
            else:
                corpus_words[stemmed_word] += 1
            class_words[data['class']].extend([stemmed_word])


def calculate_class_score(sentence, class_name, show_details=True):
    score = 0
    for word in nltk.word_tokenize(sentence):
        if stemmer.stem(word.lower()) in class_words[class_name]:
            score += (1 / corpus_words[stemmer.stem(word.lower())])
    return score
def classify(sentence):
    high_class = None
    high_score = 0
    for c in class_words.keys():
        score = calculate_class_score(sentence, c, show_details=False)
        if score > high_score:
            high_class = c
            high_score = score

    return high_class, high_score
```

```
lines = [line.rstrip('\n') for line in open('lyrics.txt')]
n=len(lines)
for i in range(n):
   sentence=lines[i]
   if(pre!=sentence):
      answer,prob=classify(sentence)
      #print(sentence)
      #print(prob)
      #print(answer)
      #speak.Speak(answer)
      if(answer!="positive"):
         sum1=sum1+1
         pos.append(lines[i])


print("positive sentences are: ",sum1)
print("total sentences are: ",len(lines))
fh=open('filter.txt','w')
for i in range(len(pos)):
   fh.write(pos[i])
   fh.write(' \n')
fh.close()
```

Model.py

```
import pronouncing
import markovify
import re
import random
import numpy as np
import os
import keras
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense

depth = 4
maxsyllables = 16
train_mode = False
artist = "kanye_west"
rap_file = "neural_rap.txt"

def create_network(depth):
      model = Sequential()
      model.add(LSTM(4, input_shape=(2, 2), return_sequences=True))
      for i in range(depth):
                  model.add(LSTM(8, return_sequences=True))
      model.add(LSTM(2, return_sequences=True))
```

```
    model.summary()
    model.compile(optimizer='rmsprop',loss='mse')

    if artist + ".rap" in os.listdir(".") and train_mode == False:
            model.load_weights(str(artist + ".rap"))
            print("loading saved network: " + str(artist) + ".rap")
    return model

def markov(text_file):
  read = open(text_file, "r").read()
  text_model = markovify.NewlineText(read)
  return text_model

def syllables(line):
    count = 0

    for word in line.split(" "):
            vowels = 'aeiouy'
            word = word.lower().strip(".:;?!")
            if word[0] in vowels:
                    count +=1
            for index in range(1,len(word)):
                    if word[index] in vowels and word[index-1] not in vowels:
                            count +=1
            if word.endswith('e'):
                    count -= 1
            if word.endswith('le'):
                    count+=1
            if count == 0:
                    count +=1
    return count / maxsyllables

def rhymeindex(lyrics):
  if str(artist) + ".rhymes" in os.listdir(".") and train_mode == False:
    print("loading saved rhymes from " + str(artist) + ".rhymes")
    return open(str(artist) + ".rhymes", "r").read().split("\n")
  else:
    rhyme_master_list = []
    print("Alright, building the list of all the rhymes")
    for i in lyrics:
      word = re.sub(r"\W+", '', i.split(" ")[-1]).lower()
      rhymeslist = pronouncing.rhymes(word)
      print(rhymeslist)
      rhymeslistends = []
      for i in rhymeslist:
        rhymeslistends.append(i[-2:])
      try:
        rhymescheme = max(set(rhymeslistends), key=rhymeslistends.count)
```

```
            except Exception:
                rhymescheme = word[-2:]
            rhyme_master_list.append(rhymescheme)
        rhyme_master_list = list(set(rhyme_master_list))
        reverselist = [x[::-1] for x in rhyme_master_list]
        reverselist = sorted(reverselist)
        rhymelist = [x[::-1] for x in reverselist]
        f = open(str(artist) + ".rhymes", "w")
        f.write("\n".join(rhymelist))
        f.close()
        return(rhymelist)

def rhyme(line, rhyme_list):
        word = re.sub(r"\W+", '', line.split(" ")[-1]).lower()
        rhymeslist = pronouncing.rhymes(word)
        rhymeslistends = []
        for i in rhymeslist:
                rhymeslistends.append(i[-2:])
        try:
                rhymescheme = max(set(rhymeslistends), key=rhymeslistends.count)
        except Exception:
                rhymescheme = word[-2:]
        try:
                float_rhyme = rhyme_list.index(rhymescheme)
                float_rhyme = float_rhyme / float(len(rhyme_list))
                return float_rhyme
        except Exception:
                return None

def split_lyrics_file(text_file):
    with open(text_file) as f:
        content = f.readlines()
    text = [x.strip() for x in content]
    return text

def generate_lyrics(text_model, text_file):
        bars = []
        last_words = []
        lyriclength = len(open(text_file).read().split("\n"))
        count = 0
        markov_model = markov(text_file)
        while len(bars) < lyriclength / 9 and count < lyriclength * 2:
                bar = markov_model.make_sentence()
                if type(bar) != type(None) and syllables(bar) < 1:
                        def get_last_word(bar):
                                last_word = bar.split(" ")[-1]
                                if last_word[-1] in "!.?,":
                                        last_word = last_word[:-1]
```

```
                        return last_word
                    last_word = get_last_word(bar)
                    if bar not in bars and last_words.count(last_word) < 3:
                        bars.append(bar)
                        last_words.append(last_word)
                        count += 1
        return bars

def build_dataset(lines, rhyme_list):
    dataset = []
    line_list = []
    for line in lines:
        line_list = [line, syllables(line), rhyme(line, rhyme_list)]
        dataset.append(line_list)

    x_data = []
    y_data = []

    for i in range(len(dataset) - 3):
        line1 = dataset[i    ][1:]
        line2 = dataset[i + 1][1:]
        line3 = dataset[i + 2][1:]
        line4 = dataset[i + 3][1:]
        x = [line1[0], line1[1], line2[0], line2[1]]
        x = np.array(x)
        x = x.reshape(2,2)
        x_data.append(x)
        y = [line3[0], line3[1], line4[0], line4[1]]
        y = np.array(y)
        y = y.reshape(2,2)
        y_data.append(y)

    x_data = np.array(x_data)
    y_data = np.array(y_data)
    print(x_data,"hy",y_data,end="\n")
    return x_data, y_data

def compose_rap(lines, rhyme_list, lyrics_file, model):
    rap_vectors = []
    human_lyrics = split_lyrics_file(lyrics_file)
    initial_index = random.choice(range(len(human_lyrics) - 1))
    initial_lines = human_lyrics[initial_index:initial_index + 2]
    starting_input = []
    for line in initial_lines:
        starting_input.append([syllables(line), rhyme(line, rhyme_list)])
    starting_vectors = model.predict(np.array([starting_input]).flatten().reshape(1, 2, 2))
    rap_vectors.append(starting_vectors)
    for i in range(100):
```

```
        rap_vectors.append(model.predict(np.array([rap_vectors[-1]]).flatten().reshape(1, 2, 2)))

    return rap_vectors

def vectors_into_song(vectors, generated_lyrics, rhyme_list):
    print("\n\n")
    print("About to write rap (this could take a moment)...")
    print("\n\n")
    def last_word_compare(rap, line2):
        penalty = 0
        for line1 in rap:
            word1 = line1.split(" ")[-1]
            word2 = line2.split(" ")[-1]
            while word1[-1] in "?!,. ":
                word1 = word1[:-1]
            while word2[-1] in "?!,. ":
                word2 = word2[:-1]
            if word1 == word2:
                penalty += 0.2
        return penalty
    def calculate_score(vector_half, syllables, rhyme, penalty):
        desired_syllables = vector_half[0]
        desired_rhyme = vector_half[1]
        desired_syllables = desired_syllables * maxsyllables
        desired_rhyme = desired_rhyme * len(rhyme_list)
        score = 1.0 - (abs((float(desired_syllables) - float(syllables))) + abs((float(desired_rhyme) -
float(rhyme)))) - penalty
        return score
    dataset = []
    for line in generated_lyrics:
        line_list = [line, syllables(line), rhyme(line, rhyme_list)]
        dataset.append(line_list)
    rap = []
    vector_halves = []
    for vector in vectors:
        vector_halves.append(list(vector[0][0]))
        vector_halves.append(list(vector[0][1]))
    for vector in vector_halves:
        scorelist = []
        for item in dataset:
            line = item[0]
            if len(rap) != 0:
                penalty = last_word_compare(rap, line)
            else:
                penalty = 0
            total_score = calculate_score(vector, item[1], item[2], penalty)
            score_entry = [line, total_score]
            scorelist.append(score_entry)
```

```
        fixed_score_list = []
        for score in scorelist:
            fixed_score_list.append(float(score[1]))
        max_score = max(fixed_score_list)
        for item in scorelist:
            if item[1] == max_score:
                rap.append(item[0])
                print(str(item[0]))
                for i in dataset:
                    if item[0] == i[0]:
                        dataset.remove(i)
                        break
                break
    return rap

def train(x_data, y_data, model):
        model.fit(np.array(x_data), np.array(y_data),batch_size=2,epochs=5,verbose=1)
        model.save_weights(artist + ".rap")

def main(depth, train_mode):
    model = create_network(depth)
    text_file = "filter.txt"
    text_model = markov(text_file)

    if train_mode == True:
        bars = split_lyrics_file(text_file)

    if train_mode == False:
        bars = generate_lyrics(text_model, text_file)

    rhyme_list = rhymeindex(bars)

    if train_mode == True:
        x_data, y_data = build_dataset(bars, rhyme_list)
        train(x_data, y_data, model)

    if train_mode == False:
        vectors = compose_rap(bars, rhyme_list, text_file, model)
        rap = vectors_into_song(vectors, bars, rhyme_list)

        f = open(rap_file, "w")
        for bar in rap:
            f.write(bar)
            f.write("\n")
main(depth, train_mode)
```

Speech.py

```
import pyttsx3
import playsound
import threading
from time import sleep
from threading import Thread

beat_to_play = r"C:\Users\DELL\Desktop\J-Component\New folder (4)\beat.mp3"

slowdown_rate = 5
intro = 10


def play_mp3(path):
    playsound.playsound(path)

def cb(name):
   print(name)

engine = pyttsx3.init()
engine.connect('started-utterance', cb)

def letters(input):
   valids = []
   for character in input:
      if character.isalpha() or character == "," or character == "'" or character == " ":
         valids.append(character)
   return ''.join(valids)

lyrics = open("neural_rap.txt").read().split("\n")
rate = engine.getProperty('rate')
engine.setProperty('rate', rate - slowdown_rate)
voices = engine.getProperty('voices')
engine.setProperty('voice','english-us')

wholesong = ""
for i in lyrics:
   wholesong = wholesong + i + " ... "


def sing(line):
   engine.say(line,name=line)
   a = engine.runAndWait()

def beat():
   play_mp3(beat_to_play)
```

```
Thread(target=beat).start()
sleep(intro)
lines = wholesong.split(" ... ")
for i in lines:
   sing(i)
```

Execution: Make three files and name them as given in the document first run Sentence classifier it will extract all the positive lines from the dataset and make a .txt file i.e. filter.txt and save the lyrics in it then run model.py in train mode it will use filter.txt as the dataset to train the model and then run that model in test mode to generate new lyrics it will save new lyrics in nural_rap.txt and this can be used by speech.py to sing the lyrics.

The various dataset can be downloaded through Kaggle.

Kayne west discography dataset
https://www.kaggle.com/viccalexander/kanyewestverses

Positive and negative words dataset
https://www.kaggle.com/harshaiitj08/positive-and-negative-words