Department of Computer Science

CPSC 3603: Project Report on NLP

Facsimile Lyric Imitator

Author:
Zyel Crier
Student ID: 900281154

Supervisor:
Dr. Rickey Lang

3.29.2022

Contents

1 Background

The goal with this project is to create new lyrics from a collection of published songs. This research touches on natural language generation (NLG) and Natural language understanding(NLU). For the NLU portion of the project, aspect-based sentiment analysis was performed on each word of the songs to help train the overall model for lyric generation. For the NLG portion of the project a Markov Chain was used to generate the new lyrics. A Markov Chain is a statistical model of a process and one of its applications is automated text generators which takes sample text like the published songs used in this project and builds a dictionary that maps sequences of words. There are many reasons why a lyric generator is a valuable tool, for example, this research could be used to create an album guessing game by generating new lyrics from an album and making users guess the album the lyric was constructed from.

2 Related Work

This research was inspired by research on automatic text summarization and sentiment analysis that myself and Mary Vu did in 2021[1]. The base principle of using frequency to determine a word's value was heavily influenced by Extractive Summarization, a method for summarizing text by identifying the most useful and necessary information in a text by extracting the most valued sentence of a text via frequency.

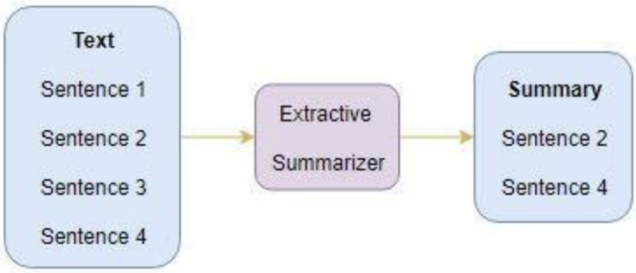

Figure 1: This flowchart depicts the Extractive summarizers flow.

It was not just enough to simply count frequencies. There was a need to identify other categories that could influence the generation of new text. So a search to identify how to extract key words was put into motion. There was the consideration of using a method for keyword extraction using clustering and distribution of nouns based on the research of Mohammad Rezaei [2] who was extracting from text specific types of words like nouns and adjectives. Ultimately that sparked a shift to finding sentiment key words. This led to finding Sentiment analysis research on twitter data [3]. in which they utilized one of NLTK'S packages to perform document-based analysis. This led to the discovery of using VADER (Valence Aware Dictionary for Sentiment Reasoning)[4] is a model used for text sentiment analysis that is

sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is available in the NLTK package and was applied directly to words to extract key sentiment words.
other works considered in this project include: Text Preprocessing using Annotated Suffix Tree with Matching Key-phrase [5],and Keyword extraction from a single document using centrality measures[6]

https://www.overleaf.com/project/624474e990eac96770e59636

3 Data-set and Features

Data was extracted using the genius API [7] the idea was to create a master Markov Chain for generation from all the songs in a single album to create a new lyric that would imitate the artist. For this project the data set originally was 17 songs from Jaden Smith's album *Syre*. However as research progressed more albums where utilized to test the results and constraints of the algorithm such as, Giveon's album *When it's All Said and Done* and Michael Jackson's Album *Thriller* among others. A considerable thing to note is the more unique words a given album has is extremely influential on the diversity of the generated lyric. As with a smaller album during generation over fitting was an issue

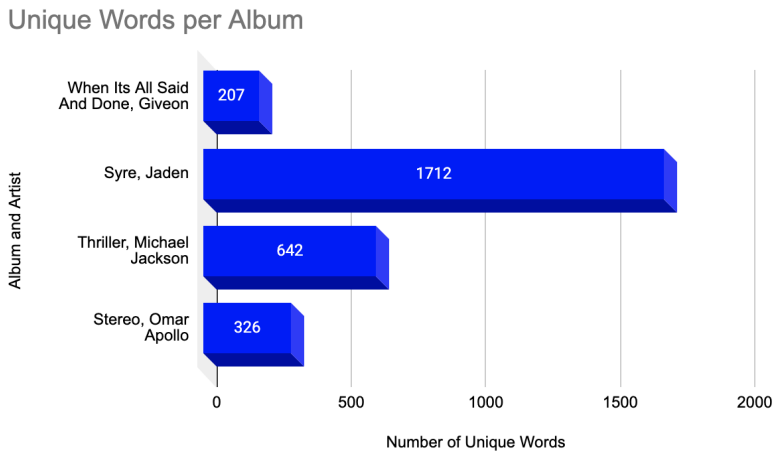

Figure 2: This graph represents the number of unique words in each album that's being utilized for the construction of new lyrics.

4 Exploratory Data Analysis

Data pre-processing for lyric generation consisted of cleaning and augmentation. To prep the song for the facsimile pipeline a script was run to get every word of a song and remove punctuation and empty spaces. This form of pre-processing increased the data we were looking at by dividing it into single words; it also allowed for the construction of the Markov Chain. Instead of removing stop words the pre-processing step was used to create a base value for them using the following equation.

The motivation for the stop word value was to obtain a non-zero value that did not eclipse all other word values. This formula is based on the occurrence's of stop words in the album going through the pipeline. Note that this formula assigns the same value to all stop words.

$$\text{StopWordValue} = \frac{\left(\frac{\sum(wf)}{(nsw)}\right)}{ns}$$

wf: is stopword frequencies. nsw: number of stopwords. ns: number of songs

5 Facsimile Model

Facsimile Model Step by Step Breakdown:

1. Input and processing
   (a) Identify stopwords for the album (do not remove them from any song but assign them ranks) This was done by using nltks's stops words collection. To identify all stop words in the album and assign a flat rank. The value assigned was done by calculating the average of all stopwords then dividing it by the number of songs in the data set.
2. Keyword Lists
   (a) **Frequency Dictionary**. In this section a master frequency table for all words in the data set was constructed. This dictionary served as the basis of all word rankings. For this project the base frequency values were calculated as document frequency meaning that instead of counting how many times a particular word showed up between all the songs values were the how many songs a particular word showed up in.

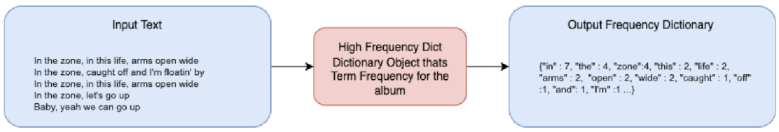

Figure 3: input output pair for Frequency Dictionary

   (b) **Key Sentiment List**. The words in this list served as a value modifier to identified sentiment keywords in the master frequency dictionary. For the implementation of this step NLTK's Vader sentiment analysis was used. Its a pre-trained model where all you have to do is plug in a token and you get a positive, negative, and neutral numerical value out. For this step tokens from the corpus were input-ed into Vader and if it had a value for either positive or neutral it was counted as a sentiment keyword and added to the

Figure 4: input output pair for sentiment list

3. Markov Chain

   (a) In this section the Markov Chain was built for the entirety of the input-ed album. The goal is to create a Markov chain that captured all states of the album. Reasoning for utilizing the chain was be able to capture how an artist structured their lyrics to achieve more natural generated lyrics by preserving it through two word prefix's and their subsequent values (the words that fallow the prefix's).

4. Lyric Generation

   (a) To create the new lyrics. Utilizing the Master Word Frequency Dictionary and the Markov Chain new lyrics are constructed. Upon construction if it is desired that sentiment be considered a Sentiment list is made in which each sentiment word in that list multiplies the corresponding word value in the Master Word Frequency Dictionary. Construction of the new lyric is simple we traverse the Markov chain favoring prefix values with the highest value given by the Master Word Frequency dictionary.

6 Result Discussion

After the successful implementation of the facsimile lyric imitator with one album I was able to create entirely new lyrics. Upon generation of new lyrics four items are required, and album and the artist, how long you want the new lyric to be and if you would like sentiment to be considered or not. Below are examples of generated lyrics based on Tina Turner's album *Wildest Dreams*. Note that all output has manually added line endings actual generated text does not its just one continuous string.

The Following lyric generated is 65 words long with sentiment being considered and a master frequency dictionary with values derived from document frequency.

> dancing in my dreams there's no hiding in the night is hot outside your window
> i hear your name in certain circles and it always makes me smile
> i spend my time has come and time
> time is not on your own but together we're strong because it's love that i feel
> his breath golden eye golden golden eye no time for love and a

The Following lyric generated is 65 words long with sentiment not considered and a master frequency dictionary with values derived from document frequency.

missing every time i got caught thief of let me go on
tell me that you love me don't you know it's for real
i know we'll go beyond this world
yeah kiss of life and the mad thing starts text in
oh you never see it coming
you just the way you are i know that something good has

The Following lyric generated is 65 words long with sentiment not considered and a master frequency dictionary with values derived from word frequency.

dancing in my heart it's getting louder all that will endure
hey now love is lost again
what u going to do
i will stand here till you see
i don't want what's yours as mine
no i'll never know the reason
but the fact is plain to see
uhuh it's the love that i might say
missing you there's a link missing from

From these examples and others performed using the attached code you can see that sentiment plays a significant role in the variety of lyrics created. It is hard to tell which option makes better lyrics as that is a subjective discernment. However with just reading these results out loud people can discern that they are lyrics or poems, which is indicative of them sounding like the natural text they are trying to imitate. There was an issue with redundancy that created loops in the lyrics solely favoring highest frequency, to mitigate this there was a degree of randomness added to the implementation.

7 Conclusion/Future Work

There are many things I would do to evolve this research in the future. With the text generation as is using the Markov Chain during construction I would build a feature for loop detection that would stop repeat phrases from occurring a certain number of times, as the algorithm as is is very prone to repetition. This could be due to lack of diversity in word values, or small amount of unique words. The research suggests its more so rooted in the construction step rather than the ranking portion. In addition to loop avoidance I would do more research into more modern methods of text generation as Markov Chains are a very well known and researched one they are not the only technique. For the future exploration of similar modern methods would be adapted to compare how they capture and imitate natural sounding output. Lastly for changes on the project as is id look into identifying other variables than sentiment that can effect the weight of words for generation as noted in the results section sentiment does make a difference in how lyrics are being made. This Research Could be used in the future to help create state-based chat bots and Ai's that can be trained on conversations to create more natural sounding responses.

References

[1] Crier Zyel and Vu Mary. Summarizing and sentiment analysis on jaden smith music. 2021.

[2] Mohammad Rezaei, Najlah Gali, and Pasi Fränti. Clrank: A method for keyword extraction from web pages using clustering and distribution of nouns. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 79–84. IEEE, 2015.

[3] Somiya Rani and Shobha Bhatt. Sentiment analysis on twitter data using support vector machine. 2016.

[4] Shihab Elbagir and Jing Yang. Twitter sentiment analysis using natural language toolkit and vader sentiment. In *Proceedings of the international multiconference of engineers and computer scientists*, volume 122, page 16, 2019.

[5] Ionia Veritawati, Ito Wasito, and T Basaruddin. Text preprocessing using annotated suffix tree with matching keyphrase. *International Journal of Electrical & Computer Engineering (2088-8708)*, 5(3), 2015.

[6] Girish Keshav Palshikar. Keyword extraction from a single document using centrality measures. In *International conference on pattern recognition and machine intelligence*, pages 503–510. Springer, 2007.

[7]

A Appendix

The contents of this section displays the implementation of the facsimile music imitator.

```
# -*- coding: utf-8 -*-
"""facsimile.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1gsiGsWjsTptwApXE0Q15U9sSfiycHnVb

# Facsimile Lyric Imitator


1.   Pre-Processing
2.   Keyword List
3.   Build Markov Model
4.   Lyric Generaton

## 1) Input & Pre-Processing

### Data Extraction
"""

!pip install git+https://github.com/johnwmillr/LyricsGenius.git

client_access_token = "SZ3jTuHWq9xKyFgcAZvDhg_wWsUOBjQtfb0mxnd3n9-zCybDEceF9oSkrEha

import lyricsgenius
LyricsGenius = lyricsgenius.Genius(client_access_token)

from bs4 import BeautifulSoup
import re
import lyricsgenius
import requests
import pandas as pd
from pprint import pprint

#regex function to clean song data
def clean_up(song_title):
    if "Ft" in song_title:
        before_ft_pattern = re.compile(".*(?=\(Ft)")
        song_title_before_ft = before_ft_pattern.search(song_title).group(0)
        clean_song_title = song_title_before_ft.strip()
        clean_song_title = clean_song_title.replace("/", "-")
```

```
    else:
        song_title_no_lyrics = song_title.replace("Lyrics", "")
        clean_song_title = song_title_no_lyrics.strip()
        clean_song_title = clean_song_title.replace("/", "-")

    return clean_song_title

#function to get songs from album
def get_all_song_titles_from_album(artist, album_name):
    artist = artist.replace(" ", "-")
    album_name = album_name.replace(" ", "-")

    response = requests.get(f"https://genius.com/albums/{artist}/{album_name}")
    html_string = response.text
    document = BeautifulSoup(html_string, "html.parser")
    song_title_tags = document.find_all("h3", attrs={"class": "chart_row-content-ti
    song_titles = [song_title.text for song_title in song_title_tags]

    clean_songs = []
    for song_title in song_titles:
        clean_song = clean_up(song_title)
        clean_songs.append(clean_song)

    return clean_songs

def clean_song(song):
    song = song.lower()
    song = re.sub('[^A-Za-z0-9-\']', ' ', song)
    song = song.replace("'","")
    return song

#download the album lyrics
def album_to_df(artist, album_name):
    data = [] #list to store all the song data

    LyricsGenius = lyricsgenius.Genius(client_access_token)  # Set up LyricsGenius
    LyricsGenius.remove_section_headers = True

    # With the function that we previously created, go to Genius.com and get all so
    clean_songs = get_all_song_titles_from_album(artist, album_name)

    for song in clean_songs:
        song_object = LyricsGenius.search_song(song, artist) #For each song in the
        if song_object != None: #If the song is not empty
          lyrics = song_object.lyrics
```

```
            lyrics = re.sub(r'\w{0,13} Lyrics\n','', lyrics)
            lyrics = re.sub(r'\d{,3}Embed','', lyrics)
            data.append({'Title' : song, 'Lyrics': lyrics})#B Lyrics\n

        else:  #If the song doesn't contain lyrics
            print('No lyrics')
    return pd.DataFrame(data, columns = ['Title', 'Lyrics']) #create and return dat

syre = album_to_df("Jaden", "Syre")

syre

"""### Pre-Processing"""

import pandas as pd
import numpy as np
import re

import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize

def get_all_words(songs):
    words = {}
    stop_words = nltk.corpus.stopwords.words('english')
    stops = {}
    for song in songs :
      song = song.lower()
      song = re.sub('[^A-Za-z0-9-\']', ' ', song)
      for line in song.split('\n'):
        for word in line.split(' '):
          if word in stop_words:
            if word not in stops:
              stops[word] = 1
            else:
              stops[word] +=1
          else:
            if len(word) == 1 and ( word != 'a' and word != 'i'):
              continue
            else:
                if word not in words:
                  words[word] = 1
                else:
                  words[word]+=1
    return words, stops
```

```
all_words, stop_vals = get_all_words(syre['Lyrics'])

print(all_words)
print(stop_vals)

print(sum(all_words.values()))
stop_word_val = round((sum(stop_vals.values())/len(stop_vals.values()))/len(syre))
print(f'value for all stop words {stop_word_val}')

"""## 2) Keyword list

### Highest Frequency
"""

def update_vals(word_vals,words_dict):
    for key, value in words_dict.items():
      if key in word_vals:
        word_vals[key] +=1
      else:
        word_vals[key] = value

import itertools

# Document Frequency So its value is based on how much it occurs in the song albums
def word_doc_freq(album):
    word_vals = {}
    stop_words = nltk.corpus.stopwords.words('english')
    for title, song in zip(album['Title'],album['Lyrics']):
      song = song.lower()
      song = re.sub('[^A-Za-z0-9-\']', ' ', song)
      for line in song.split('\n'):
        words = set(line.split(' '))
        words_dict = dict.fromkeys(words, 1)
        update_vals(word_vals,words_dict)
    return word_vals

freq = word_doc_freq(syre)

print(freq)

"""### Sentiment List"""

pip install vadersentiment

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
all_words = list(freq.keys())+ list(stop_vals.keys())
all_words

def get_senti_words(word_list):
    senti_words = set([])
    for word in word_list:
        vs = SentimentIntensityAnalyzer().polarity_scores(word)
        if (vs['pos'] >0 or  vs['neg'] >0):
          senti_words.add(word)
    return list(senti_words)

get_senti_words(all_words)

"""##3) Markov Model"""

def create_markov_chain(songs):
    chain = {} # keys (prefix,suffix) [coresponding words]
    c = 1
    for song in songs:
      chain_for_song(chain,song)
    return chain

def chain_for_song(chain,song):
    pref1 = '\n'
    pref2 = '\n'
    val = ''
    for line in song.split('\n'):
      line = re.sub('[^A-Za-z0-9-\']', ' ', song.lower())
      line = line.replace('  ', ' ')
      for word in line.split(' '):
        val = word
        key = (pref1, pref2)
        if key not in chain:
          chain[key] = {val:1}
        else:
          if word in chain[key]:
            chain[key][word]+=1
          else:
            chain[key][word] =1
        pref1 = pref2
        pref2 = val

markov_chain = create_markov_chain(syre['Lyrics'])
markov_chain
```

```
"""##4) Lyric Generation

###4a)Helper Functions
"""

def get_album():
    artist = input("Enter the name of the artist: ")
    album = input("Enter the name of the artist's album: ")
    return album_to_df(artist, album)

def get_max(songs):
    sum = 0
    for song in songs:
      sum += len(song)
    max = round(sum/len(songs))
    return  max

def get_len(album):
    max = get_max(album['Lyrics'])
    len = int(input("Enter the length of lyric you wish to create: "))
    while len > max:
      len = int(input(f"Enter the length of lyric you wish to create it must be les
    return len

def get_senti_op():
    senti = True
    choice = str(input("Would you like sentiment to be considered? (Y/N) "))
    if choice not in ['y','Y']:
      senti = False
    return senti

def get_word_value_dict(album, senti):
      all_words, stop_vals = get_all_words(album['Lyrics'])
      stop_word_val = round((sum(stop_vals.values())/len(stop_vals.values()))/len(s
      master_word_val = word_doc_freq(album)
      senti_list = []
      if senti:
        senti_list = get_senti_words(list(master_word_val.keys()))
      for word in master_word_val.items():
        if word in senti_list:
          master_word_val[word] = master_word_val[word]*2
        if word in stop_vals.keys():
          master_word_val[word] = stop_word_val
      return master_word_val

import random
```

```
def get_highest_val(p_vals,wv_dic):
    max = -1
    val = ''
    for key in p_vals.keys():
      nv = wv_dic[key]
      if nv > max:
        val = key
        max = nv
    if len(p_vals.keys()) > 3:
      val = random.choice(tuple(p_vals.keys()))
    return val

"""###4b)Main Pipeline

"""

def facsimilie_h(album, lyric_len,senti):
  word_value_dict = get_word_value_dict(album, senti)
  chain = create_markov_chain(album['Lyrics'])
  #print(word_value_dict)
  return new_lyric( word_value_dict, chain, lyric_len)

def new_lyric( wv_dic, chain,nwords):
  lyric = []
  pref1 = '\n'
  pref2 = '\n'
  val = ''
  pref = (pref1,pref2)
  while len(lyric) < nwords:
    p_vals = chain[pref]
    val = get_highest_val(p_vals,wv_dic)
    lyric.append(val)
    pref = (pref2,val)
    pref2 = val
  print(' '.join(lyric))

def facsimilie():
  album = get_album()
  lyric_len = get_len(album)
  senti = get_senti_op()
  return facsimilie_h(album, lyric_len,senti)

"""Lyric Generation Test"""

facsimilie()
```

```
facsimilie()

facsimilie()

facsimilie()

def get_word_set():
  album = get_album()
  return word_doc_freq(album)

giv = get_word_set()

jad = freq

mich = get_word_set()

omar = get_word_set()

gn = len(giv.keys())
jn = len(jad.keys())
mn = len(mich.keys())
on = len(omar.keys())
print(f"giveon: {gn}  jaden: {jn} michael: {mn} omar: {on}")
tas = {"When Its All Said And Done, Giveon": gn,"Syre, Jaden":jn,"Thriller, Michael

import matplotlib
import matplotlib.pyplot as plt

plt.bar(tas.keys(),tas.values(), width = .2, color = ['goldenrod'])
plt.xlabel("Album and Artist", fontsize = 18)
plt.ylabel("Number of Unique Words in album", fontsize = 13)
plt.title("Unique Words per Test Album", fontsize = 14)

plt.gcf().set_size_inches(18,15)
```