Hip Hop Hits Sentiments Analysis

In this project, top hip hop hits lyrics are retrieved and analysis. The list of hip hop songs are retrieved from billboard chart list. The song's details are then searched on azlyrics and if the lyrics' link is found, the scraper downloads and cleans up the lyrics. The analysis uses nltk SentimentIntensityAnalyzer to calculate the lyrics polarity.

Background

Music has a significant effect on the listener's psychological stress response and behavior. According to a research done by Myriam Thoma, Roberto La Marca, Rebecca Brönnimann, Linda Finkel, Ulrike Ehlert, and Urs Nater on The Effects of Music on the Human Stress Response, music listening impacted the psychobiological stress system. According to the research, listening to music prior to a standardized stressor predominantly affected the autonomic nervous system and to a lesser degree the endocrine and psychological stress response.

Scraping

Retrieval of data was done using python3 and python's BeautifulSoup library. The data was scraped and cleaned in python. Below is the scraping and cleanup script used. the lyrics are saved in the cache folder

```
#!/usr/bin/env python3

import os
import csv
import re

from .song import Song
from .helper import Helper
from bs4 import BeautifulSoup

# Define the global variables
CACHE_DIR = os.path.join(os.path.dirname(__file__), '..', 'cache')
LIST_SONG_DIR = "list-songs.csv"
BILLBOARD_TOP_HITS =
"http://www.billboard.com/charts/year-end/%s/hot-r-and-and-b-hip-hop-songs"
BILLBOARD_ANNUAL_MAX = 25
LYRICS_SEARCH_URL = "http://search.azlyrics.com/search.php?q=%s&w=songs&p=%d"


# Get list of top songs from billboard top chart
class Lyrics:
    link = ""

    def __init__(self, yr):
        self.year = yr
        self.link = BILLBOARD_TOP_HITS % yr

        Helper.create_dir("%s/%d" % (CACHE_DIR, self.year))
    # Generate link of the lyrics file name
    def gen_link_from_name(self):
        return "%s/%d/%s" % (CACHE_DIR, self.year, self.link.split('/')[-1])

    # Return a list of songs by scraping the billboard site
    def get_songs(self):
        page = Helper.get_page(self.link)
        if page is None:
            return

        songs = []
        soup = BeautifulSoup(page, "html.parser", from_encoding="utf-8")
        for s in soup.find_all(attrs={"class": "ye-chart__item-text"}):
            if len(songs) >= BILLBOARD_ANNUAL_MAX:
                break

            songs.append(Song(
                rank=s.find(attrs={"class":
"ye-chart__item-rank"}).get_text().strip(),
                name=s.find(attrs={"class":
"ye-chart__item-title"}).get_text().strip(),
                author=s.find(attrs={"class":
"ye-chart__item-subtitle-link"}).get_text().strip(),
                yr=self.year))

        # Save the billboard html site to file
        Helper.write_text_file("%s.html" % self.gen_link_from_name(),
soup.prettify(), 'w+')

        return songs


# Scrape song from azlyrics site
class SongScraper:

    # Constructor receives a list of songs to scrape
    def __init__(self, songs):
        if songs is None:
            self.songs = []
        else:
            self.songs = songs

    # Scrape the songs from the azlyrics site and save the lyrics
    # to file
    # cb function callback is called every time the song's lyrics
    # are scraped. it receives the textual lyrics and the song's details
    def scrape(self, cb):
        f = open(LIST_SONG_DIR, 'a+')
        writer = csv.writer(f,
                            delimiter=',',
                            lineterminator='\n',
                            quoting=csv.QUOTE_ALL)

        for song in self.songs:
            ss = self.ScrapeSongLyrics(song)
            song.polarity = cb(ss.content, song)
            writer.writerow(list(vars(song).values()))
            f.flush()
            os.fsync(f.fileno())

        f.close()

    # UpdateSong list updates the list of songs in the
    # list-song file. When the script is running, tailing
    # the file provides details on the song being scraped
    def update_song_list(self):
        with open(LIST_SONG_DIR, 'a+') as f:
            writer = csv.writer(f,
                                delimiter=',',
                                lineterminator='\n',
                                quoting=csv.QUOTE_ALL)

            for song in self.songs:
                writer.writerow(list(vars(song).values()))

    # ScrapeSongLyrics implements feature to retrieve the
    # lyrics and cleanup the data after scraping
    class ScrapeSongLyrics:
        link = ""
        content = ""
        dir_path = ""
        max_pages_scrape = 7

        def __init__(self, song):
            self.song = song

            self.dir_path = "%s/%d" % (CACHE_DIR, song.year)

            self.find_lyrics_link()
            self.get_lyrics()
            self.save_lyrics()

        # To get the lin
        def find_lyrics_link(self, pages_count=1):
            page = Helper.get_page(LYRICS_SEARCH_URL % (self.song.name,
pages_count))
            if page is None:
                return

            soup = BeautifulSoup(page, "html.parser", from_encoding="utf-8")

            counts = soup.findAll(attrs={"class": "btn btn-share btn-nav"})
            if len(counts) > 1:
                temp = int(counts[-2].get_text().strip())
                if temp < self.max_pages_scrape:
                    self.max_pages_scrape = temp

            for song in soup.find_all(attrs={"class": "visitedlyr"}):
                for s1 in song.findAll("b", text=self.song.name):
                    author = s1.parent.parent.find(text=re.compile(r'' +
self.song.author.split(' ', 1)[0] + ''))
                    if author is not None:
                        self.link = s1.parent.parent.find('a', href=True)["href"]
                        return

            pages_count += 1

            if pages_count >= self.max_pages_scrape:
                return

            self.find_lyrics_link(pages_count)

        # Get the lyrics from azlyrics site and call cleanup function to cleanup
        # the lyrics data
        def get_lyrics(self):
            print("%s %s lyrics. Link: %s" % (self.song.author, self.song.name,
self.link))
            page = Helper.get_page(self.link)
            if page is None:
                return

            soup = BeautifulSoup(page, "html.parser", from_encoding="utf-8")
            t = soup.prettify()
            t = t.split(
                "<!-- Usage of azlyrics.com content by any third-party lyrics
provider is
                prohibited by our licensing agreement. Sorry about that. -->",
                1)[-1]
            t = t.split("<!-- MxM banner -->", 1)[0]

            t1 = t.replace('<br/>', '').replace('<i>', '').replace('</i>',
'').replace('</div>', '').split('\n')

            self.content = self.consume_paren('\n'.join([line.strip() for line in
t1 if line.strip() != ""]))

        # Consume parameter consumes any text in between [] and ()
        def consume_paren(self, text):
            ret = ''
            skip1c = 0
            skip2c = 0
            for i in text:
                if i == '[':
                    skip1c += 1
                elif i == '(':
                    skip2c += 1
                elif i == ']' and skip1c > 0:
                    skip1c -= 1
                elif i == ')' and skip2c > 0:
                    skip2c -= 1
                elif skip1c == 0 and skip2c == 0:
                    ret += i
            return ret

        # Save lyrics saves the lyrics to file
        def save_lyrics(self):
            Helper.write_text_file("%s/%s-%s.txt" %
                                   (self.dir_path, self.song.name.replace("/",
"-"), self.song.author),
                                   self.content, 'w+')
```

Analysis

Analysis is done using nltk tool. Below is the analysis script.

```
#!/usr/bin/env python3
from nltk.sentiment.vader import SentimentIntensityAnalyzer

v = None


def analyze_lyrics(lyrics, song):
    sid = SentimentIntensityAnalyzer()
    polarity = sid.polarity_scores(lyrics)

    global v

    if v is None:
        v = Visualizer()

    v.visualize(polarity, song)

    return polarity


class Visualizer:
    def __init__(self):
        pass

    def visualize(self, polarity, song):
        print("Polarity: {0}, Song: {1}".format(polarity, song))

        pass
```

Running the project

The project can be ran using the make file. The command `make run` will start the project. In the command line, the details of the song and the polarity of the song are displayed after the song is scraped and analyzed. Below are the content found in the Makefile

```
run:
    @python3 src/main.py

setup:
    @pip3 install -U -r requirements.txt
```

Sample Output

Below is the sample output

```
"{'neu': 0.724, 'compound': 0.9885, 'neg': 0.058, 'pos':
0.217}","Happy","1","2014","Pharrell Williams"
"{'neu': 0.708, 'compound': 0.9924, 'neg': 0.103, 'pos': 0.189}","All Of
Me","2","2014","John Legend"
"{'neu': 0.805, 'compound': 0.9924, 'neg': 0.046, 'pos':
0.149}","Fancy","3","2014","Iggy Azalea Featuring Charli Xcx"
"{'neu': 0.814, 'compound': -0.9823, 'neg': 0.138, 'pos': 0.048}","Talk
Dirty","4","2014","Jason Derulo Featuring 2 Chainz"
"{'neu': 0.713, 'compound': 0.9872, 'neg': 0.12, 'pos': 0.167}","The
Monster","5","2014","Eminem Featuring Rihanna"
```

TO DO in the final project

1. Work on the polarity visualization
2. Improve on the polarity calculations