Audio_Similarity_Testing

November 15, 2019

1 Audio Similarity Testing

1.0.1 Zach Chase, Adam Fidler, Erik Hannesson

[ ]:
```
#Load packages
from google.colab import files
import pandas as pd
import cv2
from matplotlib import pyplot as plt
from bs4 import BeautifulSoup
import requests
import re
import time
import pickle
import numpy as np
import progressbar
import logging
import IPython.display as ipd
```

1.1 1. Data should be scraped and cleaned.

2 First Data Set - Cover Data

Note that the cover data set (Covers.list, Original.list, and cover song files) were downloaded with thanks from:

D. P. W. Ellis (2007). The "covers80" cover song data set Web resource, available: http://labrosa.ee.columbia.edu/projects/coversongs/covers80/.

Read in and Clean the Cover Data

[ ]:
```
#Read in the data
original = pd.read_csv("Original.list", sep = '/', names = ["Song", "More"])
covers = pd.read_csv("Covers.list", sep = '/', names = ['Song', "More"])

#Split original data into detailed columns
```

```
original[['Original Artist', 'Original Album', 'Original Name in File']] =␣
 ↪original.More.str.split("+", expand=True)
original = original.drop(["More"], axis = 1) #Drop column that's not needed

#Split data into detailed columns
covers[['Cover Artist', 'Cover Album', 'Cover Name in File']] = covers.More.str.
 ↪split("+", expand = True)
covers = covers.drop(["More"], axis = 1) #Drop column that's not needed

#Clean original df - Convert "_" into " "
original["Song"] = original.Song.replace("_", " ", regex = True)
original["Original Artist"] = original["Original Artist"].replace("_", " ",␣
 ↪regex = True)
original["Original Album"] = original["Original Album"].replace("_", " ", regex␣
 ↪= True)

#Clean cover df - Convert "_" into " "
covers["Song"] = covers.Song.replace("_", " ", regex = True)
covers["Cover Artist"] = covers["Cover Artist"].replace("_", " ", regex = True)
covers["Cover Album"] = covers["Cover Album"].replace("_", " ", regex = True)

#Add .wav to end of name in files
original["Original Name in File"] = original["Original Name in File"] + ".wav"
covers["Cover Name in File"] = covers["Cover Name in File"] + ".wav"
```

Merge and Display Data

[ ]:
```
music = pd.merge(original, covers, on = "Song")
music.head()
```

[ ]:
```
                       Song  ...                Cover Name in File
0    A Whiter Shade Of Pale  ...     2-A_Whiter_Shade_Of_Pale.wav
1                Abracadabra  ...                11-Abracadabra.wav
2          Addicted To Love  ...           09-Addicted_To_Love.wav
3  All Along The Watchtower  ...  15-All_Along_The_Watchtower.wav
4    All Tomorrow s Parties  ...    06-All_Tomorrow_s_Parties.wav

[5 rows x 7 columns]
```

[ ]:
```
#Display image of .wav files in folder
#FIX

img = cv2.imread('folder_image.png', 0)/255

plt.imshow(img, cmap = 'gray')
plt.figure(figsize = (20,20))
plt.rcParams['figure.figsize'] = 200, 160
plt.show()
```

3 Second Data Set - Lawsuit Data

Code to scrape data from the website https://blogs.law.gwu.edu/mcir/cases/

[ ]:
```
# configure logging
logging.basicConfig(filename='scrape.log', filemode='a+',
                    format='%(levelname)s - %(message)s')

def scrape_link_info(row, case, verbose=False):
    """
    Scrapes artist/song/link-to-audio info for a given case. Given the row␣
 ↪object
    for a given case, this finds the link to the specific case page, requests
    the html and obtains the artist(s)/song(s)/link(s)-to-audio data.

    Parameters:
        row (BeautifulSoup object): soup object for the row of the case in
            question.
        case (str): case name. This name will appear as the first entry of each
            row and is the basis upon which to join the two dataframes.
        verbose (bool): theoretically adds print statements to check what's␣
 ↪going
            on, but in actuality this currently does nothing.

    Returns:
        df (pandas dataframe): constructs a pandas dataframe of the scraped
            content. Columns are formatted as:

             ------------------------------------------------------------
            | case-name | complaining | artist | title | link-to-audio |
            |-----------+-------------+--------+-------+---------------|
            |    str    |     bool    |  str   |  str  |      str      |
            |___________|_____________|________|_______|_______________|

            There is no index column.
    """
    # get link to page with audio
    link = row.find(class_='column-3')
    link = link.find(href=re.compile(r'http')).get('href')

    # request page that has audio links on it
    audio_page = requests.get(link).text
    soup = BeautifulSoup(audio_page, 'html.parser')

    try:
        # get complaining artist/defending artist columns
        comp = soup.find_all(class_='ms-rteTableEvenCol-default')[1] # only␣
 ↪second
```

```
        defn = soup.find_all(class_='ms-rteTableOddCol-default')[1] # entry␣
↪needed
    except IndexError:
        # if no second entry, case doesn't have requisite information
        return None

    # get data from each complaining/defending artist columns
    comp_data = comp.find_all(class_='ms-rteElement-P')
    defn_data = defn.find_all(class_='ms-rteElement-P')

    # lists to hold cleaned complaining/defending artist/title/link info
    clean_data = []
    clean_temp = []
    temp_link = ''

    entry_num = 0       # tracks artist/title/link groups
    comp_bool = False   # tracks if we are in the comp_data or defn_data set

    for d_group in [comp_data, defn_data]:
        # true when in comp_data iter, false when in defn_data iter
        comp_bool = not comp_bool
        clean_temp = [] # this should already be empty, but *just* in case
        group_len = len(d_group) # to track when we should append link and␣
↪break
        end_group = False        # binary switch to indicate end of group

        for item in d_group:
            if len(clean_temp) == 0:
                # add case title
                clean_temp.append(case)
                # add bool to track if this is complaining/defending data
                clean_temp.append(comp_bool)

            try:
                # if this contains the hyperlink, get it
                href_temp = item.find(href=re.compile(r'http'))

                if re.match('Hear Sound Recording', href_temp.text):
                    # if this link contains some sort of audio data, get link
                    cur_link = href_temp.get('href')
                    clean_temp.append(cur_link)
                    clean_data.append(clean_temp)
                    clean_temp = []
                    entry_num = -1

                    # if temp_link:
```

```
                    #     # if we've already collect a link, append with comma␣
↪sep
                    #     temp_link += ',' + cur_link
                    # else:
                    #     temp_link += cur_link

            except AttributeError:
                # if this isn't something with a hyperlink, collect its data
                if entry_num == 0 or entry_num == 1:
                    text_temp = item.text

                # check if this is the '' entry between artist/title/link␣
↪groups
                    # note that that is a *long* dash
                if text_temp == '' or text_temp == '\n':
                    # if we are here, then this is the end of the current␣
↪sub-group
                    if len(clean_temp) == 4:
                        clean_temp.append(None)
                        clean_data.append(clean_temp)
                        clean_temp = []
                        entry_num = -1
                else:
                    # if this is artist/title data, add it
                    clean_temp.append(text_temp)
            finally:
                entry_num += 1

            # if end_group:
            #     clean_temp.append(temp_link)
            #     clean_data.append(clean_temp)
            #     clean_temp = []
            #     temp_link = ''
            #     entry_num = -1


    # now convert/return the data in clean_data into a pandas dataframe
    columns = ['case', 'complaining', 'artist', 'title', 'link']
    return pd.DataFrame(clean_data, columns=columns)

def base_scrape(base_url='https://blogs.law.gwu.edu/mcir/cases/', link_df=None,
                gen_df=None, save=True, wait_time=1, verbose=False,␣
↪v_verbose=False):
    """
    """
    gen_columns = ['year', 'country', 'case', 'complaining_work',
                   'defending_work', 'complaining_author', 'defending_author']
```

```
    link_columns = ['case', 'complaining', 'artist', 'title', 'link']

    if link_df is None:
        # if we didn't pass in a link_df to add to, initialize empty one
        link_df = pd.DataFrame(columns=link_columns)
    if gen_df is None:
        # if we didn't pass in a gen_df to update, initialize an empty one
        gen_df = pd.DataFrame(columns=gen_columns)

    # request source
    source = requests.get(base_url).text
    # soup object
    soup = BeautifulSoup(source, 'html.parser')
    # get the table of stuff
    row_hover = soup.find(class_='row-hover')
    # grab each row
    rows = row_hover.find_all(class_=re.compile(r'row-[\d]+ (even|odd)'))

    if verbose:
        # if we want to know where we're at, use a progressbar
        rows = progressbar.progressbar(rows)

    # go through each row and get the information
    for row in rows:
        # pause for a moment before moving on
        time.sleep(wait_time)
        # get the columns that contain what we want
        cols = row.find_all(class_=re.compile(r'column-[\d]+'))

        # try to extract information, if anything breaks log what broke and␣
↪skip
        try:
            # extract all the information, filter to what we want
            info = [item.text for item in cols]
            info = info[:3] + info[4:]

            # pause for a moment, then update link dataframe
            link_df_temp = scrape_link_info(row, info[2], verbose=v_verbose)

            if link_df_temp is None:
                continue

        except KeyboardInterrupt as e:
            raise e

        except Exception as e:
            logging.exception("Exception occured")
```

```
            continue

        # if everything went well, update our dataframes
        link_df = link_df.append(link_df_temp)
        gen_df = gen_df.append(pd.DataFrame([info], columns=gen_columns))

        if v_verbose:
            print(link_df)
            print('\n')
            print(gen_df)
            print('\n\n')

    # save the data
    if save:
        with open('data/gen_df', 'wb') as f:
            pickle.dump(gen_df, f)
        with open('data/link_df', 'wb') as f:
            pickle.dump(link_df, f)

    return gen_df, link_df


def scrape_audio():
    pass
```

Display Lawsuit Data

[ ]:
```
gen = pd.read_csv('gen_df.csv')
gen.loc[:, ~gen.columns.str.contains('^Unnamed')]
```

[ ]:
```
     year  ...                                    defending_author
0    1845  ...                                       Samuel Carusi
1    1887  ...                                     W.D.Hendrickson
2    1914  ...                                        Joseph James
3    1915  ...                                        Jeff Godfrey
4    1916  ...                                      Al Piantadosi
..    ...  ...                                                 ...
194  2018  ...  Marriott International, Inc, and Universal Pic...
195  2018  ...                                        Cyndi Lauper
196  2018  ...                  Gwen Stefani & Pharrell Williams
197  2019  ...   Eric Avalos, Courtney Rico, Chavalos Music, Inc.
198  2019  ...                                    Nice for what

[199 rows x 7 columns]
```

[ ]:
```
links = pd.read_csv('gen_df.csv')
links.loc[:, ~links.columns.str.contains('^Unnamed')]
```

[ ]:
```
     year  ...                                    defending_author
0    1845  ...                                       Samuel Carusi
```

```
1    1887  ...                                        W.D.Hendrickson
2    1914  ...                                           Joseph James
3    1915  ...                                           Jeff Godfrey
4    1916  ...                                         Al Piantadosi
..    ...  ...                                                    ...
194  2018  ...  Marriott International, Inc, and Universal Pic...
195  2018  ...                                           Cyndi Lauper
196  2018  ...                    Gwen Stefani & Pharrell Williams
197  2019  ...   Eric Avalos, Courtney Rico, Chavalos Music, Inc.
198  2019  ...                                       Nice for what

[199 rows x 7 columns]
```

Download Song Files from Lawsuit

[ ]:
```
import pickle
import pandas as pd
import requests
import time
import re
import sys

# commandline argument to indicate starting point
n = int(sys.argv[1])

# read pickled data
with open('./data/link_df','rb') as f:
    df = pickle.load(f)

# iterate through each link and download sound file
for i, link in enumerate(df['link'][n:]):
    time.sleep(1)

    try:
        # download sound file
        audio = requests.get(link)

        # set filename
        filename = link[len(re.sub(r'/[^/]*\.(mp3|wav|wma)','',link))+1:]
        print(filename,end=' ')

    except:
        # log download errors
        with open('./link_log.txt','+a') as f:
            f.write('Error getting accessing {}th link\n'.format(n+i))
    try:
        # save sound file
        with open('./audio_files/' + filename, '+wb') as f:
```

```
            f.write(audio.content)

            # progress tracker
            print('{0}/{1}'.format(i+1,len(df['link'][n:])))
    except:
        # log save errors
        with open('./dl_log.txt','+a') as f:
            f.write('Error reading/saving {}\n'.format(link))
```

Audio Samples

[42]:
```
print('Joe Satriani, "If I Could Fly"')
ipd.Audio('satriani_fly-21ehlgn.wav')
```

```
Joe Satriani, "If I Could Fly"
```

[42]:
```
<IPython.lib.display.Audio object>
```

[43]:
```
print('Coldplay, "Viva la Vida')
ipd.Audio('satriani_viva-2a493s5.wav')
```

```
Coldplay, "Viva la Vida
```

[43]:
```
<IPython.lib.display.Audio object>
```

3.1 2. Identify potential problems with the data:

a) *Evaluate the source — is it reliable or not? why? Evaluate potential biases and other problems with the data (e.g. political data gathered by a particular party trying to get a particular result. Or positive pharmaceutical data from the company selling the pharmaceutical) What will you do to deal with these problems?*

The Music Copyright Infringement Resource (MCIR) is a collection of over a century's worth of musical copyright cases. The resource was established in 1997 by Charles Cronin, JD, Ph.D., and musician from Los Angeles, the current director of the project. According to the MCIR, "musical experts, on behalf of litigants and courts, analyze works in music copyright infringement disputes," and cases are submitted by attorneys and scholars in the music industry on a regular basis, with the most recent being Michael Skidmore v. Led Zeppelin in July 2019. Each case includes the name, number, and date of the case, the complaining and defending works, applicable media, and comments by Cronin et al. and occasional opinions from the ruling judges. While these comments and opinions are subject to bias, they are not considered in our analysis.

---

b) *Are there missing results or other things that look wrong? why? what will you do to deal with these? Values that are out of range (e.g. percentages not in 0-100, or counts that are negative or much bigger than physically possible) must be identified and removed or corrected.*

Some cases have multiple complaining or defending works, which complicate direct one-to-one comparisons, and some have no audio clips whatsoever. The cases with no audio are disregarded, as they cannot be analyzed. The court rulings of each case are of particular interest, as we hope to compare the rulings with our determined metric of "closeness" between complaining and defending songs. However, the court rulings are not apparent in each case listed in MCIR and will require some text processing. Furthermore, the court cases are inconsistent in the link text to listen to each song snippet, which further complicates audio data collection.

3.2 3. Evaluate the suitability of your data for answering the questions in the proposal. If it is not suitable, adjust the questions or acquire new data as necessary.

The data gathered is sufficient to answer the questions of the proposal. Our original question is to compare similar/dissimilar songs through DFT to create a metric that will determine if how similar the two songs are to each other. Using this metric, we will compare songs in music copyright infringement cases to determine if there is a mathematical way to determine the strength of the case. We will first compare songs in the cover data set to determine the similarities/differences between songs. We will then apply our knowledge to the lawsuit data set to answer our questions.

3.3 4. Revise any other aspects of your proposal in light of what you have learned from examining the data.

The data we have collected so far is conducive to our initial proposal. We have yet to obtain a parody dataset, but given the cover dataset it may be unnecessary. We will reevaluate this necessity as we perform the DFT analysis.