ADS 509 Module 3: Group Comparison

Renetta Nelson

May 29, 2023

The task of comparing two groups of text is fundamental to textual analysis. There are innumerable applications: survey respondents from different segments of customers, speeches by different political parties, words used in Tweets by different constituencies, etc. In this assignment you will build code to effect comparisons between groups of text data, using the ideas learned in reading and lecture.

This assignment asks you to analyze the lyrics and Twitter descriptions for the two artists you selected in Module 1. If the results from that pull were not to your liking, you are welcome to use the zipped data from the "Assignment Materials" section. Specifically, you are asked to do the following:

- Read in the data, normalize the text, and tokenize it. When you tokenize your Twitter descriptions, keep hashtags and emojis in your token set.
- Calculate descriptive statistics on the two sets of lyrics and compare the results.
- For each of the four corpora, find the words that are unique to that corpus.
- Build word clouds for all four corpora.

Each one of the analyses has a section dedicated to it below. Before beginning the analysis there is a section for you to read in the data and do your cleaning (tokenization and normalization).

In [ ]:
```
import os
import re
import emoji
import pandas as pd

from collections import Counter, defaultdict
from nltk.corpus import stopwords
from string import punctuation
from wordcloud import WordCloud

from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer
```

In [ ]:
```
# Use this space for any additional import statements you need

import PIL
```

In [ ]:
```
# Place any addtional functions or constants you need here.

# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison
tw_punct = punctuation - {"#"}

# Stopwords
sw = stopwords.words("english")

# Two useful regex
whitespace_pattern = re.compile(r"\s+")
hashtag_pattern = re.compile(r"^#[0-9a-zA-Z]+")

# It's handy to have a full set of emojis
all_language_emojis = set()

for country in emoji.EMOJI_DATA :
    for em in emoji.EMOJI_DATA[country] :
        all_language_emojis.add(em)

# and now our functions
def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity, and num_tokens most common
        tokens. Return a list of
    """
```

```
    # Place your Module 2 solution here

    num_characters = 0
    for i in tokens:
        num_characters = num_characters + len(i)


    # Fill in the correct values here.
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = num_unique_tokens / num_tokens
    num_characters = num_characters #len(list(tokens)) #(nltk.FreqDist(nltk.Text(tokens))).N()

    if verbose :
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")

        # print the five most common tokens

    return([num_tokens, num_unique_tokens,
            lexical_diversity,
            num_characters])



def contains_emoji(s):

    s = str(s)
    emojis = [ch for ch in s if emoji.is_emoji(ch)]

    return(len(emojis) > 0)


def remove_stop(tokens) :
    # modify this function to remove stopwords

    #return (t for t in tokens if t.lower() not in stopwords)
    return(tokens)

def remove_punctuation(text, punct_set=tw_punct) :
    return("".join([ch for ch in text if ch not in punct_set]))

def tokenize(text) :
```

```
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter. """

    # modify this function to return tokens
    #for i in text:
    #    text = i.split(" ")

    return(text)

def prepare(text, pipeline) :
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

Data Ingestion

Use this section to ingest your data into the data structures you plan to use. Typically this will be a dictionary or a pandas DataFrame.

In [ ]:
```
# Feel fre to use the below cells as an example or read in the data in a way you prefer

data_location = "C:/Users/nelso/Desktop/" # change to your location if it is not in the same directory as your notebook
twitter_folder = "twitter/"
lyrics_folder = "lyrics/"

lyric_artist1 = "cher/"

artist_files = {'cher':'cher_followers_data.txt',
                'robyn':'robynkonichiwa_followers_data.txt'}
```

In [ ]:
```
twitter_data = pd.read_csv(data_location + twitter_folder + artist_files['cher'],
                           sep="\t",
                           quoting=3)

twitter_data['artist'] = "cher"
```

In [ ]:
```
twitter_data_2 = pd.read_csv(data_location + twitter_folder + artist_files['robyn'],
                             sep="\t",
                             quoting=3)
twitter_data_2['artist'] = "robyn"
```

```
twitter_data = pd.concat([
    twitter_data,twitter_data_2])

del(twitter_data_2)
```

In [ ]:
```
# read in the lyrics here

# Create dictionary for lyrics data
lyrics_data = pd.DataFrame()


artist_names = []
song_names = []
lyrics_songs =[]


for artists in os.listdir(data_location + lyrics_folder):
    for lyrics in os.listdir(data_location + lyrics_folder + artists):
        artist, song_name = lyrics.split("_")

        song_name = song_name.replace(".txt", " ")
        artist_names.append(artist)
        song_names.append(song_name)


        with open(data_location + lyrics_folder + artists + '/' + lyrics) as infile:
            next(infile)
            next(infile)
            next(infile)
            next(infile)

            lyrics_songs.append(infile.read())


lyrics_data["artist"] = artist_names
lyrics_data["s_names"] = song_names
lyrics_data["lyrics"] = lyrics_songs
```

Tokenization and Normalization

In this next section, tokenize and normalize your data. We recommend the following cleaning.

Lyrics

- Remove song titles
- Casefold to lowercase
- Remove stopwords (optional)
- Remove punctuation
- Split on whitespace

Removal of stopwords is up to you. Your descriptive statistic comparison will be different if you include stopwords, though TF-IDF should still find interesting features for you. Note that we remove stopwords before removing punctuation because the stopword set includes punctuation.

Twitter Descriptions

- Casefold to lowercase
- Remove stopwords
- Remove punctuation other than emojis or hashtags
- Split on whitespace

Removing stopwords seems sensible for the Twitter description data. Remember to leave in emojis and hashtags, since you analyze those.

In [ ]:
```
# apply the `pipeline` techniques from BTAP Ch 1 or 5

my_pipeline = [str.lower, remove_punctuation, tokenize, remove_stop]

lyrics_data["tokens"] = lyrics_data["lyrics"].apply(prepare,pipeline=my_pipeline)
lyrics_data["num_tokens"] = lyrics_data["tokens"].map(len)

twitter_data["tokens"] = twitter_data["description"].apply(prepare,pipeline=my_pipeline)
twitter_data["num_tokens"] = twitter_data["tokens"].map(len)
```

In [ ]:
```
twitter_data['has_emoji'] = twitter_data["description"].apply(contains_emoji)
```

Let's take a quick look at some descriptions with emojis.

In [ ]:
```
twitter_data[twitter_data.has_emoji].sample(10)[["artist","description","tokens"]]
```

Out[ ]:

	artist	description	tokens
557474	cher	Events Producer 🦫 Ngaarr #Gomeroi #Dhanggati ...	events producer 🦫 ngaarr #gomeroi #dhanggati ...
458758	cher	sempre bella, Inamorata della vita 💃🕺 felice😍💞...	sempre bella inamorata della vita 💃🕺 felice😍💞 ...
3969388	cher	✨ Creativo MX Mexicano 🌎 Monterrey, NL.	✨ creativo MX mexicano 🌎 monterrey nl
651934	cher	🙃	🙃
18997	robyn	copywriter + independent beckett scholar, @gol...	copywriter independent beckett scholar goldsm...
3691420	cher	Mother of one beautiful 6 year old girl, Jaidi...	mother of one beautiful 6 year old girl jaidin...
992573	cher	Aries ♈	aries ♈
462421	cher	she/her 🤺✨ pm of tls @ cloudflare... major ...	sheher 🤺✨ pm of tls cloudflare major key a...
3720594	cher	Paulista/Paulistano em Carapicuíba/SP BR 2.7 🎉...	paulistapaulistano em carapicuíbasp BR 27 🎉 vi...
420789	cher	26♈ Ele/Dele 🏳️‍🌈 💭 💤 🎶 💻 🎬 🎮	26♈ eledele 🏳️‍🌈 💭 💤 🎶 💻 🎬 🎮

With the data processed, we can now start work on the assignment questions.

Q: What is one area of improvement to your tokenization that you could theoretically carry out? (No need to actually do it; let's not make perfect the enemy of good enough.)

A: For the Twitter data, a Tweet tokenizer could be carried out. By implementing this tokenizer, emojis can be considered different words as well.

Calculate descriptive statistics on the two sets of lyrics and compare the results.

In [ ]:

```
# your code here

print("Descriptive Statistics for Cher Lyrics\n")

descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'cher'], verbose=True)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'cher'], verbose=False)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'cher'], verbose=False)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'cher'], verbose=False)

print("\n\n")
```

```
print("Descriptive Statistics for Robyn Lyrics\n")


descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'robyn'], verbose=True)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'robyn'], verbose=False)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'robyn'], verbose=False)
descriptive_stats(lyrics_data["lyrics"].loc[lyrics_data['artist'] == 'robyn'], verbose=False)
```

```
Descriptive Statistics for Cher Lyrics

There are 316 tokens in the data.
There are 316 unique tokens in the data.
There are 341088 characters in the data.
The lexical diversity is 1.000 in the data.



Descriptive Statistics for Robyn Lyrics

There are 104 tokens in the data.
There are 95 unique tokens in the data.
There are 145840 characters in the data.
The lexical diversity is 0.913 in the data.
```

Out[ ]:
```
[104, 95, 0.9134615384615384, 145840]
```

Q: what observations do you make about these data?

A: The first thing I noticed was the difference between the tokens and unique tokens for each set of lyrics. The tokens and unique tokens are equal for the cher lyrics; however for the robyn lyrics you see a slight difference in the amount. There are more characters in the cher lyrics than in the robyn lyrics. Due to these differences, the lexical diversity is different as well.

Find tokens uniquely related to a corpus

Typically we would use TF-IDF to find unique tokens in documents. Unfortunately, we either have too few documents (if we view each data source as a single document) or too many (if we view each description as a separate document). In the latter case, our problem will be that descriptions tend to be short, so our matrix would be too sparse to support analysis.

To avoid these problems, we will create a custom statistic to identify words that are uniquely related to each corpus. The idea is to find words that occur often in one corpus and infrequently in the other(s). Since corpora can be of different lengths, we will focus on the

concentration of tokens within a corpus. "Concentration" is simply the count of the token divided by the total corpus length. For instance, if a corpus had length 100,000 and a word appeared 1,000 times, then the concentration would be $\frac{1000}{100000} = 0.01$. If the same token had a concentration of 0.005 in another corpus, then the concentration ratio would be $\frac{0.01}{0.005} = 2$. Very rare words can easily create infinite ratios, so you will also add a cutoff to your code so that a token must appear at least n times for you to return it.

An example of these calculations can be found in this spreadsheet. Please don't hesitate to ask questions if this is confusing.

In this section find 10 tokens for each of your four corpora that meet the following criteria:

1. The token appears at least `n` times in all corpora
2. The tokens are in the top 10 for the highest ratio of appearances in a given corpora vs appearances in other corpora.

You will choose a cutoff for yourself based on the side of the corpus you're working with. If you're working with the Robyn-Cher corpora provided, `n=5` seems to perform reasonably well.

In [ ]:
```
# your code here

new_token1 = []


#Initialize corporas

corpora1 = lyrics_data.loc[lyrics_data['artist'] == 'cher']
corpora2 = lyrics_data.loc[lyrics_data['artist'] == 'robyn']



#Count total number in corpora

tot_corp1 = len(corpora1)
#print("Total Corpus Length (Corpora 1): ", tot_corp1)

tot_corp2 = len(corpora2)
#print("Total Corpus Length (Corpora 2): ", tot_corp2)


#def get_ratio(word, fd_corpus_1, fd_corpus_2, len_1, len_2):
#   frac_1 = get_word_frac (word, corpora1, tot_corp1)
#   frac_2 = get_Word_frac(word, corpora2, tot_corp2)#

#   if fac_2 > 0:
```

```
#       return(frac_1 / frac_2)

#   else: return(float('NaN'))


def count_words(df, column='tokens', preprocess=None, min_freq=2):

    # process tokens and update counter
    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into data frame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)


corp1_count = lyrics_data["num_tokens"].loc[lyrics_data['artist'] == 'cher']
corp2_count = lyrics_data["num_tokens"].loc[lyrics_data['artist'] == 'robyn']




corp1 = count_words(corpora1)
print("Corpora 1", corp1.head(5))
corp2 = count_words(corpora2)
print("Corpora 2", corp2.head(5))
```

```
Corpora 1         freq
token
        58913
e       31202
o       25328
t       21654
a       19322
Corpora 2         freq
token
        24515
e       12523
o       10394
t       10113
i        8262
```

Q: What are some observations about the top tokens? Do you notice any interesting items on the list?

A: I was not able to run the code using the examples in the book. I kept getting an error. I was able to look through the lyric data using the definition provided below. I noticed that three common tokens were e, t, and o. These had the highest frequencies for lyrics of both cher and robyn. I found blanks on the list. I am not sure if I forgot to account for something or if the blanks were supposed to be there. So I found that pretty interesting.

Build word clouds for all four corpora.

For building wordclouds, we'll follow exactly the code of the text. The code in this section can be found here. If you haven't already, you should absolutely clone the repository that accompanies the book.

In [ ]:
```
from matplotlib import pyplot as plt

def wordcloud(word_freq, title=None, max_words=200, stopwords=None):

    wc = WordCloud(width=800, height=400,
                   background_color= "black", colormap="Paired",
                   max_font_size=150, max_words=max_words)

    # convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq
```

```
    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    plt.title(title)

    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")


def count_words(df, column='tokens', preprocess=None, min_freq=2):

    # process tokens and update counter
    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into data frame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)
```

In [ ]:

```
corpora1 = lyrics_data.loc[lyrics_data['artist'] == 'cher']
corpora2 = lyrics_data.loc[lyrics_data['artist'] == 'robyn']
corpora3 = twitter_data.loc[twitter_data['artist'] == 'cher']
corpora4 = twitter_data.loc[twitter_data['artist'] == 'robyn']


corp1 = count_words(corpora1)
corp2 = count_words(corpora2)
corp3 = count_words(corpora3)
corp4 = count_words(corpora4)



word_corp1 = wordcloud(corp1["freq"])
print("Word Cloud for Corpora 1 \n")
```

```
print(word_corp1)


word_corp2 = wordcloud(corp2)
print("\n Word Cloud for Corpora 2 \n")
print(word_corp2)


word_corp3 = wordcloud(corp3['freq'])
print("\n Word Cloud for Corpora 3 \n")
print(word_corp3)


word_corp4 = wordcloud(corp4['freq'])
print("\n Word Cloud for Corpora 4 \n")
print(word_corp4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[180], line 14
      9 corp3 = count_words(corpora3)
     10 corp4 = count_words(corpora4)
---> 14 word_corp1 = wordcloud(corp1["freq"])
     15 print("Word Cloud for Corpora 1 \n")
     16 print(word_corp1)

Cell In[171], line 19, in wordcloud(word_freq, title, max_words, stopwords)
     16 if stopwords is not None:
     17     counter = {token:freq for (token, freq) in counter.items()
     18                           if token not in stopwords}
---> 19 wc.generate_from_frequencies(counter)
     21 plt.title(title)
     23 plt.imshow(wc, interpolation='bilinear')

File c:\Users\nelso\AppData\Local\Programs\Python\Python38\lib\site-packages\wordcloud\wordcloud.py:508, in WordCloud.g
enerate_from_frequencies(self, frequencies, max_font_size)
    505 transposed_font = ImageFont.TransposedFont(
    506     font, orientation=orientation)
    507 # get size of resulting text
--> 508 box_size = draw.textbbox((0, 0), word, font=transposed_font, anchor="lt")
    509 # find possible places using integral image:
    510 result = occupancy.sample_position(box_size[3] + self.margin,
    511                                    box_size[2] + self.margin,
    512                                    random_state)

File c:\Users\nelso\AppData\Local\Programs\Python\Python38\lib\site-packages\PIL\ImageDraw.py:671, in ImageDraw.textbbo
x(self, xy, text, font, anchor, spacing, align, direction, features, language, stroke_width, embedded_color)
    669     font = self.getfont()
    670 if not isinstance(font, ImageFont.FreeTypeFont):
--> 671     raise ValueError("Only supported for TrueType fonts")
    672 mode = "RGBA" if embedded_color else self.fontmode
    673 bbox = font.getbbox(
    674     text, mode, direction, features, language, stroke_width, anchor
    675 )

ValueError: Only supported for TrueType fonts
```

Q: What observations do you have about these (relatively straightforward) wordclouds?

A: I tried executing this code; however, I keep getting an error. I have researched trying to figure out how to fix it but I found nothing. I made sure I had the updated library installed as well. I am still getting this error.