CS 410 Final Project - Classifying Song Genre Based on Lyrics

Project Documentation

1. Our code consists of a web scraper to gather song data (including the artist, genre, and lyrics), a Naive Bayes classifier to predict the genre of a song (based on the input of its lyrics), and a web GUI for the user to input lyrics and receive a genre judgment.
2. We'll begin with our web scraper code, which we ran out of a Jupyter Notebook.

```
!pip install spotipy
import spotipy
import json
import requests
import re
import html
import os
import time

from spotipy.oauth2 import SpotifyClientCredentials
from bs4 import BeautifulSoup
```

The first cell consists of our imports - we made use of a variety of networking and systems libraries to scrape lyrics from the web and store them in a text file. We used the Spotipy package to make working with the Spotify API easy, and we used BeautifulSoup for our HTML parsing and extraction.

```
client_id = '3ce1f5cf8ebf4d33aef320f9c4834037'
client_secret = '94476eff9d0a410a85b7793d24a11c83'

client_credentials_manager = SpotifyClientCredentials(client_id, client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

#[R&B -> Usher, Rap -> Eminem, Rock -> Pink Floyd, Country -> Morgan Wallen, Jaz
# Gospel -> Kirk Franklin, Funk -> James Brown]
initial_genre_artist = {'R&B': ('Usher','23zg3TcAtWQy7J6upgbUnj'), 'Hip-Hop': ('

#genre -> artist IDs -> songs
song_info = {}
```

We began by connecting to the Spotify API using our client ID and client secret. We selected an artist to begin with from each genre (R&B, Hip-Hop, Rock, Country, Jazz, K-pop, Reggae, Gospel, and Funk).

```
for genre, artist in initial_genre_artist.items():

    related_artist_data = sp.artist_related_artists(artist[1])['artists']
    song_info[genre] = {}
    song_info[genre][artist] = set()

    for rel in related_artist_data:
        song_info[genre][(rel['name'],rel['id'])] = set()

count = 0
for genre, artists in song_info.items():
    for (name, a_id), songs in artists.items():
        albums = sp.artist_albums(a_id, limit=5)['items']

        for ind_album in albums:
            album_id = ind_album['id']
            tracks = sp.album_tracks(album_id, limit=10)['items']
            for track in tracks:
                songs.add(track['name'])
                count += 1
```

For each artist, we pulled the list of their albums and then pulled each song title from each album. Then, we pulled a list of each artist's related artists to continue gathering song titles for each of our selected genres.

```
songs_per_genre = 100
failed_song_limit = 10

fl = open('lyrics_train_data.txt', 'a')

current_artist_found = False
current_song_found = False

for genre in song_info:

    song_quota_reached = False
    genre_song_count = 0
    songs_genre = ['', genre]

    artist_info = song_info[genre]

    for (name, a_id), songs in artist_info.items():

        current_fail_count = 0
```

We then began looping through our song titles to pull their lyrics - we decided to gather a certain amount of songs per genre to prevent the scraper from running for too long, and we also set a certain limit for the amount of songs that could "fail" - this covered situations in which AZLyrics did not have the lyrics for the song or in which we submitted an invalid URL. If this limit was hit while we were scraping through a certain artist's songs, the scraper would move to the next artist.

```
for song in songs:

    artist_name = name.split(' (')[0]
    artist_name = re.sub(r'[^A-Za-z0-9 ]+', '', artist_name)
    artist_name = artist_name.replace('  ', '')
    artist_name = artist_name.replace(' ', '')
    artist_name = artist_name.lower()

    song_name = song.split(' (')[0]
    song_name = re.sub(r'[^A-Za-z0-9 ]+', '', song_name)
    song_name = song_name.replace('  ', '')
    song_name = song_name.replace(' ', '')
    song_name = song_name.lower()

    azlyrics_url = 'http://www.azlyrics.com/lyrics/' + artist_name + '/' + song_name + '.html'
    print(azlyrics_url + ' ...')

    payload = {'api_key': 'b5bcd3b27a08ef14160145e2170fffa1', 'url': azlyrics_url}
    page = requests.get('http://api.scraperapi.com', params=payload)
    soup = BeautifulSoup(page.content, "html.parser")

    lyrics = soup.find('div', class_='ringtone')
```

For each song, we transformed the artist name and song title into the URL structure that AZLyrics uses - we did this through a combination of string processing and regular expressions. We then created the AZLyrics URL for that specific song, and submitted a request for the HTML through the scraping API we were using. Once we received the HTML, we created a BeautifulSoup object from it and looked for the closest tag - ringtone - to the div containing the song lyrics.

```
if lyrics == None:
    #artist is probably not found in AZ Lyrics
    current_fail_count += 1
    if current_fail_count >= failed_song_limit:
        break

    print('Not found')
    continue
else:
    current_fail_count = 0

lyrics = lyrics.findNext('div')
lyrics = str(lyrics)
lyrics = lyrics.split('-->')[1]
lyrics = lyrics.replace('<br/>', '')
lyrics = lyrics.replace('</div>', '')
lyrics = lyrics.replace('\n', ' ')

if len(lyrics) != 0:
    fl.write(lyrics + '\n')
    fl.flush()
```

If we're unable to find the lyrics div on the page, we consider this a "failed" song and we count it towards the limit. Otherwise, we strip the lyrics div of its tags as well as other HTML decorators, and write it to our data text file.

Once our data was gathered, we needed to feed it into our model for training. We first needed to store the data in a local variable.

```
def get_train_data(path):
    genres = ['R&B', 'Hip-Hop', 'Rock', 'Country', 'Jazz', 'K-Pop', 'Reggae', 'Gospel', 'Funk']
    with open(path, 'r') as fl:
        all_lines = fl.readlines()
        lyrics = ''
        training_data = []
        count = 0
        for line in all_lines:
            text_wo_newl = line.strip('\n')

            if text_wo_newl in genres:
                training_data.append([lyrics, text_wo_newl])
                lyrics = ''
                print('Genre: {}, Count: {}'.format(text_wo_newl, count))
                count = 0
            elif text_wo_newl != '':
                count += 1
                lyrics += text_wo_newl + ' '

        return training_data
```

The method that does this is get_train_data, which takes in a path to the file storing our raw data. We parse our raw data file and store the data in a 2D array where the first column has the lyrics (in a bag of words manner) for all the songs parsed in a genre, and the second column contains the according genre. Our file was stored in a way where each song in a given genre had its lyrics separated by a new line, and at the end the genre is printed. Thus, we read until we saw a genre, and then stored all the previously seen lyrics as lyrics for that genre, and appended that to the training_data array.

After having the data stored in our data structure, we then send it into the Naive Bayes Classifier.

```
#train Naive Bayes Classifier to predict genre based on lyrics
def train_naive_bayes(input_data):
    #Perform stop word removal, punctuation removal, stemming, on each row
    count_vec = CountVectorizer()
    tf_idf_trans = TfidfTransformer()
    stemmer = PorterStemmer()

    np_input_data = np.array(input_data)
    for train_row in np_input_data:
        text = train_row[0]
        text_tokens = word_tokenize(text)
        tokens_without_sw = [word for word in text_tokens if not word in stopwords.words('english') and not word in string.punctuation]
        stemmed_tokens = [stemmer.stem(word) for word in tokens_without_sw]
        cleaned_lyrics = ' '.join(stemmed_tokens)
        train_row[0] = cleaned_lyrics
```

The method to do this was train_naive_bayes, which took in the 2D array from get_train_data. We first cleaned the data by removing stop words and punctuation, and then we stemmed the words for each genre.

```
clf = GaussianNB()

train_lyrics = np_input_data[:,0]
# labels below map to -> ['R&B', 'Hip-Hop', 'Rock', 'Country', 'Jazz', 'K-Pop', 'Reggae', 'Gospel', 'Funk']
train_genre = [0, 1, 2, 3, 4, 5, 6, 7, 8]
genre_term_count = count_vec.fit_transform(train_lyrics)
term_count_tfidf = tf_idf_trans.fit_transform(genre_term_count)

clf.fit(term_count_tfidf.toarray(), train_genre)

#save Naive Bayes model
pickle.dump(clf, open(classifier_file, 'wb'))
pickle.dump(count_vec, open(cv_file, 'wb'))
pickle.dump(tf_idf_trans, open(tfidf_file, 'wb'))
```

We then convert the data into a form which can be passed into the Gaussian Naive Bayes classifier (this code segment is also part of the train_nave_bayes function). We want to vectorize each genre's lyrics. We start by getting the term frequency of words for each genre with the sklearn CountVectorizer's fit_transform method. We then take this vector and perform a TF-IDF transformation on it with the sklearn TfidfTransformer's fit_transform method. We use fit_transform so that our transformer instances can use the same vocabulary/model on our test data later. Lastly, we pass the data and labels into the classifier and fit it. We pickle the models so that we can use them later for test data.

To perform a prediction, we used the trained Naive Bayes model referenced above.

```
def predict_genre(lyrics):
    count_vec = pickle.load(open(cv_file, 'rb'))
    tf_idf_trans = pickle.load(open(tfidf_file, 'rb'))
    stemmer = PorterStemmer()

    for text in lyrics:
        text_tokens = word_tokenize(text)
        tokens_without_sw = [word for word in text_tokens if not word in stopwords.words('english') and not word in string.punctuation]
        stemmed_tokens = [stemmer.stem(word) for word in tokens_without_sw]
        cleaned_lyrics = ' '.join(stemmed_tokens)
        text = cleaned_lyrics

    test_lyrics_cv = count_vec.transform(lyrics)
    tf_idf_test_lyrics = tf_idf_trans.transform(test_lyrics_cv)

    #load Naive Bayes model
    nb_classifier = pickle.load(open(classifier_file, 'rb'))
    predicted = nb_classifier.predict(tf_idf_test_lyrics.toarray())
    return label_mapping[predicted[0]]
```

The method that does this is predict_genre, which takes in a string for the lyrics. We loaded in the CountVectorizer, TfIdfTransformer, and GaussianNB models that were created in train_nave_bayes. We then perform the same data cleaning done in train_nave_bayes and convert it into a usable vector format with the CountVectorizer and TfIdfTransformer. This way the data is in the same format as the training. We then use the classifier and call predict to get the predicted label. The variable label_mapping is an array that stores the genres, where each index of the array maps to the according prediction output.

Our last component was our GUI. This was written in HTML using Bootstrap, and Javascript for dynamic rendering. For the HTML, we have a textarea to paste the lyrics of a song in, a button, which when clicked will send the request to predict the genre, and a text input box to fill with this predicted genre.

```
<script>

    document.getElementById("submit").onclick = async function() {
        var lyrics = {
            "lyrics": document.getElementById("enter_lyrics").value
        };

        axios.post('http://localhost:5000/get_genre', data={"lyrics":lyrics})
        .then(function (response) {
            document.getElementById("genre_spec").value = response.data['result'];
        })

    };
</script>
```

When the button is clicked, the following function is run. It takes the lyrics from the textarea and sends a post request (using axios) and passes the lyrics within the request. On response, it fills the text input box with the result.

3. In order to use our GUI, first run pip install -r requirements.txt to install the necessary dependencies, and then run python lyrics_entry_ui.py to start up the Flask server. Open entry.html and enter in any set of song lyrics and the classifier will judge its genre! In order to run the web scraper, you'll need to sign up for both the Spotify API (https://developer.spotify.com/documentation/web-api/) and the Scraper API (https://www.scraperapi.com/). Input your Spotify Client ID and Client Secret where it's requested by the Spotipy object and input your Scraper API key into the HTML request. Then, simply run all the cells to gather your song data!
4. Work Breakdown
    a. Avinash Nathan (anathan4) - web scraper, Naive Bayes classifier, web GUI

b. Aaron Aftab (aaronaa2) - web scraper, Spotify/Scraper API, data cleaning and string processing