Vibify

By Sonic Searchers

We're *Sonic Searchers*, the creators of

Vibify

Vibify is a NLP tool to help songwriters, lyricists, and the music-passionate get data on lyrics

About_Sonic_Searchers

Aleks is a senior majoring in Computer Science at Queens College with a focus in Data Science/ML, graduating December 2023, and is also a former professional musician

Stephen is a senior at John Jay College studying Applied Math and Computer Science. Loves building nlp models and hopes to pursue a career in data science

Deepankar is a Junior at CCNY, studying CS. Has a passion for applying data driven solution to problems. Want to apply Machine Learning knowledge to build something cool.

Motivation

Use Case:

You need to write a “happy sounding” song for a commercial, but aren't sure how “happy” it sounds?

Give your lyrics to Vibify and it will return a rating from 0 (very sad) to 1 (very happy)!

Motivation 🔊

Use Case:

Do you have writer's block and are looking for inspiration?

Input the lyrics you have written so far, and Vibify returns the 5 most similar songs to yours.

Content inspiration!

Vibify's insights (output) includes:

- Positive/Negative Sentiment Analysis (*valence*)
- The genre of the lyrics (Pop, Rock, R&B, Country, etc.)
- Top-5 Lyrically Similar Songs

#_Motivation

In the beginning, we asked 2 questions:

Can you predict whether a song sounds positive or negative from its *lyrics alone?*

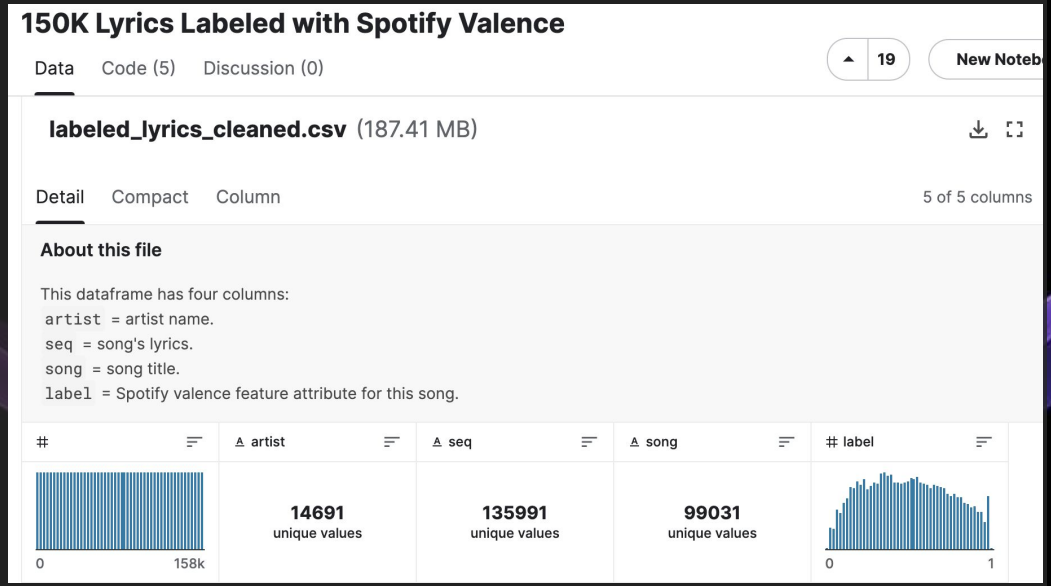
Can you predict the genre of the song from *lyrics alone?*

(i.e. do some lyrics show up more frequently in certain genres than others?)

Model Training (input)

Our data set had published songs with *valence* labels + we added genre labels by web scraping Genius.com

…but what is “*valence*”?

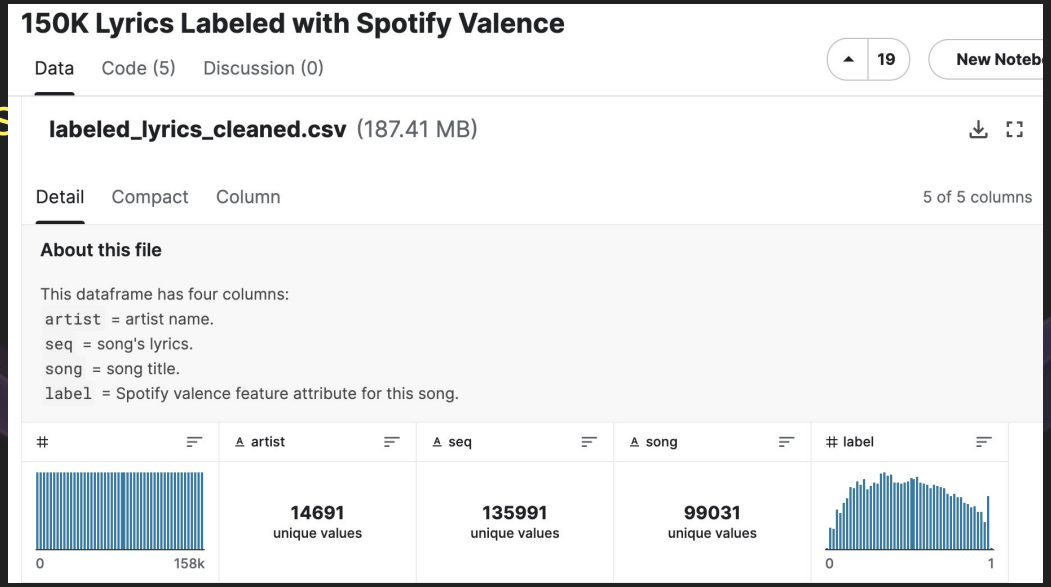


#_Project_Description

Dataset

1. songs with *valence* labels
2. genre labels scraped from Genius.com (Pop, Rock, R&B, Country, Rap)

…but what is "*valence*"?



The Spotify Web API defines it as:

Valence (a decimal score from 0 to 1):

Songs with...

- high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with
- low valence sound more negative (e.g. sad, depressed, angry)

Our Tech Stack

- Python
- Pandas
- Numpy
- Scikit-Learn
- Beautiful Soup
- Matplotlib

- NLTK
- TensorFlow
- Keras
- Word2vec
- BERT
- Streamlit

Our Models/Techniques

- RNN using LSTM for Genre and Valence Prediction
- Cosine Similarity for Top-5 Similar Songs

#_Why_LSTM? (Long Short Term Memory)

- Rnn are good at analysing sequential data, like words
- However, Rnn faces the vanishing gradient problem
- LSTM is a special version of RNN
- LSTM solves this problem

Deep: how to explain RNN(Recurrent Neural Network) and LSTM in easy words?

ChatGTP:

A recurrent neural network (RNN) is a type of neural network that is able to process sequential data, such as text, audio, or time series data. This is achieved by introducing feedback connections, which allow information to flow from previous steps in the sequence to the current step. In other words, an RNN is able to "remember" information from previous steps, which makes it well-suited for tasks such as language modeling and machine translation.

An LSTM (long short-term memory) network is a specific type of RNN that is able to learn long-term dependencies in data. This is accomplished using gates within the network, which control the flow of information and allow the network to learn which information to remember and which to forget. LSTMs are often used for tasks such as language translation and text generation.

In summary, an RNN is a type of neural network that can process sequential data, and an LSTM is a specific type of RNN that is able to learn long-term dependencies in data.

OUR ML PROCESS

#_Cleaning_lyrics

```
def text_processing_pipeline(a_string):
    a_string = make_lowercase(a_string)
    a_string = remove_punc(a_string)
    a_string = remove_stopwords(a_string)
    a_string = lem_words(a_string)
    return a_string
```

Vectorize X

```
# Limiting our tokenizers vocab size to 10000
max_words = 10000


# create the tokenizer
tokenizer = Tokenizer(num_words=max_words)


# Fit the tokenizer
tokenizer.fit_on_texts(X)


# Create the sequences for each sentence, basically turning each word into its index position
sequences = tokenizer.texts_to_sequences(X)


index_word = tokenizer.index_word


# # Limiting our sequencer to only include 300 words
max_length = 300


# # Convert the sequences to all be the same length of 300
X = pad_sequences(sequences, maxlen=max_length, padding='post')
print(X.shape)
```

```
(158353, 300)
```

Building LSTM nueral net

```
# This creates the Neural Network
model = Sequential()

# This embedding layer basically will automatically create the word2vec vectors based on your text data.
model.add( Embedding(max_words, 32, input_length=max_length) )
model.add(LSTM(50,return_sequences=True,dropout =0.2))
model.add(LSTM(50,dropout =0.2))
model.add(Dense(1,kernel_initializer='normal',  activation='linear'))
optimizer = optimizers.Adam(lr=0.003)
model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mse'])
```

LSTM_valence_model

Training

```
callback = callbacks.EarlyStopping(monitor='val_mse',patience = 2,restore_best_weights=True)
hist = model.fit(X_train, y_train,
                 validation_split=0.2,
                 epochs=15, batch_size=20,callbacks=[callback])
```

Testing

```
mse= model.evaluate(X_test, y_test, verbose=0)[1]

print('Test mse with stacked LSTM:', mse)
```

```
Test mse with stacked LSTM: 0.04978620260953903
```

LSTM_Genre_model

```
#Create LSTM model
from tensorflow.keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Embedding(max_words, 32, input_length=max_length))

model.add(layers.LSTM(64, dropout=0.1,  return_sequences=True))
model.add(layers.LSTM(64, dropout=0.1))
model.add(layers.Dense(5, activation='softmax'))

model.summary()
```

#_Challenges

Challenges met:

- Class Imbalances
- Misclassification of genres (pop and rock were often confused to be the same)
- Creating the right architecture for our neural net

#_Next_Steps 🔊

What's Next for Vibify?

- Expanded dataset = better predictions
- More inputs/filters
  - ex: Input lyrics & genre -> get similar songs from **that** genre

- *Next Level:* Audio analysis
  - ability to extract lyrics
  - ability to analyze pos/neg sentiment from the sound of the audio

Thank you!

Questions? | Feedback?