

LINE OF ENQUIRY

- Using the 'Design for a context' pathway's prompt 'analysing the context from a sound perspective', I wanted to develop an algorithm that translates the text from a book to a piece of music, quantitatively. For this, I needed to look at discrete values that could serve as building blocks to the final music piece and I decided to focus on the various emotions in the book. My primary line of enquiry now became thus:
- As a story progresses, can we extract and analyse the primary emotions of the story?
- Can this 'emotional makeup' of a story be turned into a piece of music? How fast, and more importantly, how accurately can this be done?

For the purposes of discussion within this document, I will be using 'A Thousand Splendid Suns' by Khaled Hosseini as my example text. This book is a personal favourite of mine but more importantly, I feel it works well with the dataset used for extracting emotions and is also a rich ground for analysing a variety of primary emotions.

INTRODUCTION

Sentiment analysis is the process by which we analyse text (whether it is in the form of tweets, movie reviews, essays, prose) in terms of its judgments, responses as well as feelings. This is being extensively used in fields like data mining, web mining, and social media analytics because sentiments are the most essential characteristics to judge human behaviour. Writing carries meaning and on a large, dynamic scale, it is not possible to evaluate the content of these writings manually. Such a thing would be infeasible, inefficient and a waste of time and resources. Sentiment analysis works on discovering opinions, classify the emotion they convey, and categorize them based on pre-specified divisions. While it is used to make such business decisions and gauge user feedback, it can just as well be used to visualize long-form text such as stories and novels, although the methodology is different. For this, the story's text is cleaned for processing, their individual words read, 'emotions' selected, classified, and finally, this dataset can be visualized.

In this project, I took this analysis one step further and worked on developing a process to 'musify' it. This is not the same as data sonification, as the process will reveal. The resulting set of programs allow the user to go from text to a finished piece of music in less than a minute and is usually accurate in conveying the meaning.

BACKGROUND RESEARCH

The idea of sonifying data is far from new. To quote, “There is an increasing need for sonification of big data, and such a need is understandable in a world with increasingly complex information; in an era of big data, humans long for ‘additional [cognitive] bandwidth’ (Scaletti and Craig, 1991, p.10) to comprehend their surroundings. “ (Fillela, 2018) [1]

Numbers by themselves are often meaningless. They have the tendency to grow into proportions that are sometimes much beyond what the human ‘bandwidth’ of understanding is. This is where sonification comes in. While it might not have a direct application or utility, it is successful in attracting the attention of an otherwise uninterested audience; this is, after all, one of the requirements of a good data visualization. Humans understand sound without as much help as they would need in other mediums. Infographics can be confusing, graphs can be hard to understand but the human mind’s penchant for spotting patterns and understanding sonic structures makes sound one of the best mediums to convey large information. NASA does it all the time, notably in their [sonification of the sun’s atmosphere](#) or [lunar exploration](#). This is done by assigning a set of rules to be followed as a particular item in the dataset is encountered. For example, again from NASA’s sonification of lunar exploration:

Every musical instrument in “Giant Leaps” is propelled by this data. Here’s a breakdown:

The pitch of the notes conveys the amount of scientific activity associated with the Moon in each year. The higher the pitch, the more scientific publications in that year.

- *Strings and electric piano = scientific activity associated with the Apollo program*
- *Brass instruments = scientific activity associated with all other lunar missions*

The percussion instruments indicate the passage of time.

- *Clock ticking = months*
- *Snare drum = years*
- *Bass drum = decades*
- *Cymbals crashing = launches*

Each fragment of sound in the musical piece can be mapped back to the data it is drawn from. This seemed easy enough and in its simplest form, all one had to do was assign a particular frequency of sound to each unique data item and it too could form a musical piece.

This proved a little more difficult when dealing with something like stories. Since this data is not self-evident, that is, it is not there in numeric form, we need to find a way that can give us numerical values to work with from the text. One way of looking at these long pieces of text was to isolate keywords that conveyed a particular kind of emotion. This is still an unreliable exercise because “the technology is still developing and it can be unpredictable when dealing with short sentences, but it has been shown to be reliable when drawing conclusions from large amounts of text (Dodds and Danforth, 2010; Pang and Lee, 2008).” So while not completely accurate, the out-of-context word associations can be used effectively, and often cancel out the errors when worked on with large texts. The ‘signal’ that is received from the image is strong enough to be used meaningfully. For this, I used the NRC Emotion Lexicon.

NRC EMOTION LEXICON

The NRC Emotion Lexicon was created by crowdsourcing to Amazon's Mechanical Turk, Mohammad and Yang, 2011; Mohammad and Turney, 2010). The methodology followed for collating this database of over 14,000 words was as follows:

1. A word is presented in form of a question and the closest possible association is given in one of the multiple choices (mixed with three other useless terms). For example:

Q.1 What is the word shark is closest related to?

1. Fish 2. Car 3. Olive 4. Tree

2. The participant chooses the closest association.
3. This association is reviewed by 5 different people.
4. Based on a majority vote, it is decided if the word is correctly associated to the relevant emotion.

Thus, while it is not perfectly correct, there is a good chance that the words in the lexicon are correctly corresponding to the emotion they've been categorized under. This, coupled with the assumption that these few keywords are good enough indicators of the overall larger flow of text, provides a good way to obtain the aforementioned discrete values that we needed to look for.

The resulting database is a table of 1s and 0s, with '1' being affirmative for that category.

English (en)	Positive	Negative	Anger	Anticipation	Disgust	Fear	Joy	Sadness	Surprise	Trust
aback	0	0	0	0	0	0	0	0	0	0
abacus	0	0	0	0	0	0	0	0	0	1
abandon	0	1	0	0	0	1	0	1	0	0
abandoned	0	1	1	0	0	1	0	1	0	0
aberration	0	1	0	0	1	0	0	0	0	0

METHODOLOGY: TEXT PROCESSING

In order to explain the text processing and classification, I will be breaking down the final code into the steps I took to write it.

The first order of business for any text processing exercise, for best results, is the removal of 'bad characters'. These are punctuation marks, numbers and special characters which might alter the word from its base form (and something that the Emotion Lexicon may not have). To do this, I wrote a script that reads in the text file character by character and matches the input with a predefined list of bad characters. In the eventuality that a bad character is encountered, it is replaced with whitespace instead.

```
%23Take User input for file name. Must be saved to working directory
name = input("Enter file name:")
%23Clean the text for use
filename = "%s.txt" %name

%23 initializing bad_chars_list
bad_chars = [';', ':', '!', "*", '.', ',', '-', '_', '(', ')', '"', '1', '2', '3', '4', '5',
'6', '7', '8', '9', '0']

%23 Read in file
with open(filename, encoding="utf8") as infile:
    contents = infile.read()

%23 using replace() to remove bad_chars
for i in bad_chars:
    contents = contents.replace(i, '')
```

Now, the text file is overwritten with this new version, which is devoid of any stray characters that might disrupt the analysis. We now read this file into the program again, except we shall do so word-by-word. But before I did that, I converted the Emotion Lexicon into an array of strings. These arrays can be long, with each array containing at least 3000 terms. When we read in the words from the text file, these can be matched to a pair in the array. For this exercise, I used only four emotions.

```
anger = ['abandoned', 'abandonment', 'abhor', 'abhorrent', 'abolish',....]
joy = ['absolution', 'abundance', .....]
sadness = ['abandoned', ....]
trust = ['example word']
```

With the text cleaned and the arrays defined, we are now ready to match the words to their arrays. Here we are faced with a small problem of deciding which emotions we will give more importance to. The words in the Lexicon can correspond to multiple emotions; it can come under both 'Joy' and 'Surprise', for example. Since these are 'elif' statements, encountering the first instance of an emotion will immediately negate all subsequent matches. Each emotion was also given a corresponding identifier. I classified my order of emotions as such:

1. Joy
2. Sadness
3. Anger
4. Fear

The resulting code to match the words to these emotions, by order of preference, looked like this.

```
%23 Python program to read text word by word

DNA = []          %23declare DNA list
wordlist = []     %23declare word list
lim = 0           %23declare limit variable to

%23Read in text word by word and see if it falls in any one of these categories. Limit to first 50 entries.
while lim <= 50:
    for word in contents.split():

        if word in joy:
            DNA.append('1');
            wordlist.append(word)
            lim = lim+1

        elif word in sadness:
            DNA.append('2');
            wordlist.append(word)
            lim = lim+1

        elif word in anger:
            DNA.append('3');
            wordlist.append(word)
            lim = lim+1

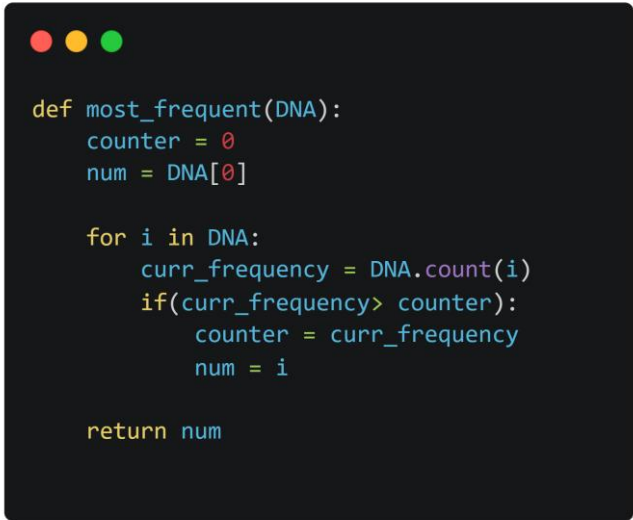
        elif word in fear:
            DNA.append('4');
            wordlist.append(word)
            lim = lim+1
```

The data output this produced was similar to this:

```
dna = [1,2,2,1,1,1,1,1,1,1,1,4,1,3,1,3,2,3,2,1,1,2,3,1,3,1]
```

This is a time-series dataset, since the stream is a result of reading the progression of emotions in the book. This can be refined further to ensure easier file handling. I wrote a script that converted this to a dataframe and saved that as a CSV file. Additionally, another textfile containing the data stream was saved in the same directory and automatically copied to the user's clipboard.

One last important information we could use from this stream was enumerating the most frequently occurring emotion, since this would be a good indicator of the overall emotion of the piece of text.



```
def most_frequent(DNA):  
    counter = 0  
    num = DNA[0]  
  
    for i in DNA:  
        curr_frequency = DNA.count(i)  
        if(curr_frequency > counter):  
            counter = curr_frequency  
            num = i  
  
    return num
```

This is all we have to do in Python for now. The text has been processed, the emotions extracted and the output stream of data saved in all possible useful formats. The next hurdle was figuring out rules to convert these numbers into a musical piece.

METHODOLOGY: MUSIFICATION

In the process of looking for ways to make music with code, I came across a dedicated music coding program called Sonic Pi. This was written in Ruby and was a live coding program, built in with samples, synths and special effects which could allow me to produce something very similar to NASA's Lunar Exploration musical piece. For this, I wrote down some basic rules of conversion that I would follow:

1. In order to depict the overall emotion of the text, I could vary the tempo, or beats per minute (BPM), of the final music piece. This meant that an overall sad sample of text would be played at 30 BPM, while something happier would be played at 80 BPM.

2. A background looping score could be varied depending on the overall emotion too. Dark ambient music would correspond to sadness or fear and something upbeat would correspond to happier texts.
3. The pitch of the musical instrument could be varied with the type of emotion encountered.

In week three, I began experimenting with it and playing random sounds that I could use to make my own output but this proved difficult for a number of reasons. Firstly, the software was intuitive but geared towards DJ-style music creation. I did not intend to create anything of the sort. Secondly, although I had a set of rules in mind, none of them seemed to produce a musical output neither coherent enough nor really very pleasant to listen to. The latter is something I gave significant importance to, since the text could get long and the resulting music piece also had to sound 'good' to engage the user.

I realized that I had jumped into music creation without formally learning about the rules that governed its structure first. This meant I was wasting time generating cacophony instead of real music. I decided to read a little more on the history of generative and algorithmic music, and this is when I came across Mozart's Dice Game.

MOZART'S DICE GAME

Mozart's Dice Game is one of the earliest attempts at algorithmic music. The premise was simple; Mozart pre-composed 176 bars of music and arranged them in a matrix lookup table. All the player had to do was roll a set of dice, note the sum of the results and then look up the corresponding bar from the lookup table. Repeating these 16 times, he or she could 'generate' a waltz within minutes and chances are, it would be original. And because these bars had been written with random arrangements in mind, any order of them could sound good. This worked well for me! One of my very basic requirements was, as mentioned, that the music had to sound good while having been generated from the data. The process is effective since nearly infinite number of waltzes can be generated, and each one has the distinct sound that is attributed to Mozart. One of the drawbacks I realized later was that even though the piece was unique in the literal sense, it sounded incredibly similar to something generated from a different output. This means, of course, that wildly different will ever be produced, not matter what the values we choose. This is mainly because Mozart has composed the phrases to follow a strict harmonic structure since this is the easiest way to guarantee that each generation will sound correct.

The pros outweighed the cons in every respect, and I decided to take the process forward.

Instead of figuring out the logic from scratch, I decided to build on top of an existing program written to simulate this game by developer Robin Newman. The only difference here was that replaced his dice-throwing logic with my logic of reading in the datastreams and looking up the corresponding bars from the lookup table.

In essence, the code was taking the data and performing a multiplication on it (to make it a bit larger and look up something further up the table).

```
#Read in 50 entries form the DNA array
0.upto(50) do |i|
  dna.each { |a|
    d = dna[a]
    dl << d #
    cl << w[d][i] #add looked up bar number to list
  }
end
puts "Emotional DNA "
puts dl #print dice totals
puts "List of bars from lookup table"
puts cl #print bar choices

#ln left hand notes, ld left hand durations
#rn right hand notes, rd righthand durations
ln[0]=[:g2,:b3,:g3,:fs3,:e3]
ld[0]=[q,sq,sq,sq,sq]
rn[1]=[:f5,:d5,:g5]
ln[1]=[:f3,:d3,:g3]
rn[2]=[:a4,:fs4,:g4,:b4,:g5]
```

Now, the output was both pleasant sounding and reasonably related to the data it was mapped from.

VALIDATION

While the process had been refined and worked smoothly, I wanted to know how well users understood the piece for what it is. I chose a really small test group; Madhavi, Anshuman and Parvathy, and asked them the following set of questions. This was based on the soundscape I generated from Chapter 1 of A Thousand Splendid Suns, which I've uploaded here:

<https://soundcloud.com/aman-bhargava-528794340/dna>

Their answers follow.

ARE YOU ABLE TO GAUGE WHAT THE OVERALL MOOD IS?

1. Yes, the clip sounds happy.
2. The clip sounds happy.
3. The overall emotion seems to be happiness.

This was correct, the most frequent emotion was indeed joy.

CAN YOU DETECT A MUSICAL STRUCTURE? REPETITIONS AND SO ON?

1. No, it just sounds good.
2. Yes, there are repeating notes.
3. Yes, there are repeating bars.

IF I TOLD YOU THAT EACH OF THOSE BARS CORRESPOND TO A PARTICULAR EMOTION, DOES THAT SEEM REASONABLE TO UNDERSTAND?

1. No, they just sound like notes.
2. No.
3. No, these are just notes.

SUMMARY

While the repeating structure is discernible, and the overall mood is clear, the participants could not draw a connection between the repetitions and the data associated with it. This meant that while it is musically pleasing, the notes in the final output cannot be understood to correspond to the emotion they are mapped from. They were essentially just reduced to notes and had been abstracted so far away from what they had been generated from that it no longer had any resemblance. This might also have something to do with the wrong choice of data. Such a stream of data, which is unpredictable, might not possess the qualities of being 'sonified'. I had taken liberties in making this sound pleasing and had therefore made the choice between making it sound good or making it scientifically accurate. This is therefore, finally an act of designing rules and designing an algorithm wherein data is ascribed artistic aesthetics. However, since the users are able to gauge the overall mood of the piece it is successful in transmitting information to the listener as well as existing as a self-contained piece of music.

CONCLUDING NOTES

This process demanded a lot of planning, reasoning, reading up on music theory and figuring out logic problems but in the end, given what the users have responded with, it seems to be that I might be engaging in 'conversion hysteria'. This means that I might be looking to convert things into music that really might not make sense and where the final piece is veiled in ambiguity. While the entire process is now taking less than a minute to complete, it is yet to produce any particular utility or application (in my opinion) but serves in generating interest from users about the particular text it is generated from. However, a combination of musification and audio visualisation to create a product with both sonic and visual facets might result in an 'application' which I seem to be seeking. My future exploration in to this should explore potential frontend, accessible software developments which could simplify the musification process, even if it is not entirely accurate, that lets users handle text to music conversions with ease.

REFERENCES

1. From Once Upon a Time to Happily Ever After: Tracking Emotions in Novels and Fairy Tales Saif Mohammad. <https://www.aclweb.org/anthology/W11-1514.pdf>
2. Data Sonification in Creative Practice: <https://pearl.plymouth.ac.uk/bitstream/handle/10026.1/14583/2019BonetFilla10504498PhD.pdf?isAllowed=y&sequence=1>
3. Robin Newman's Mozart Dice Game: <https://rbnrpi.wordpress.com/project-list/mozart-dice-generated-waltz-revisited-with-sonic-pi/>
4. Mozart - Musical Game in C K. 516f* <http://www.asahi-net.or.jp/~rb5h-ngc/e/k516f.htm>
5. Davis, N. (2016). If you could listen to dark matter, just what would it sound like? The Observer, 14 February 2016.