Machine Learning Engineer Nanodegree

Capstone Project

Music Information Retrieval & Emotion Annotation using Lyrics

Project Overview

Music Information Retrieval (MIR)[1] is the science of retrieving information from music. The information extracted using MIR techniques usually involves the beats per minute, key, pitch, etc, and this extraction is based on the ***sound*** of the music. The extracted information can be used for applications such as **Acoustic Fingerprinting**[2] (identifying the currently playing song, a feature that apps like MusixMatch, Shazam, SoundHound, Google Assistant, etc. have), and also for classifying music into genres or emotions. The latter of which can be referred to as **Emotion Annotation**.

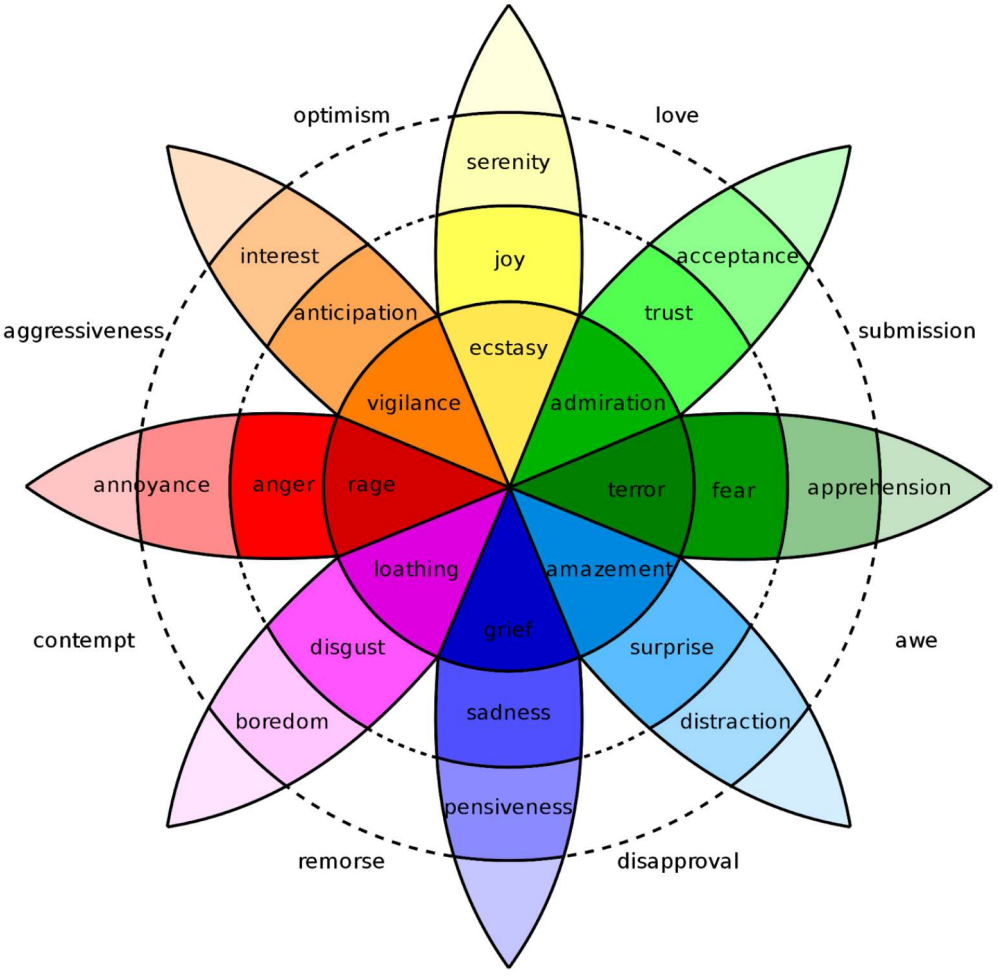
Lyrics are also an important aspect of music (well, unless it's instrumental). While it may be difficult, if not impossible, to use lyrics to retrieve information such as the key, pitch, or the beats per minute, lyrics may be able to help us understand the **meaning** of a song, or the message that it is trying to convey. With that information, we may be able to understand what emotions a song is trying to express (not elicit).

Note that there's a significant difference between the two. A person may have first listened to ***Happy by Pharrell Williams***[3] at a difficult time and the song may not necessarily **elicit** joy from him/her, even though the song's lyrics itself could be said to **express** joy.

In this project, I am attempting to **cluster** songs into multiple categories/topics based on the words present in their lyrics using a **Topic Model**[4]. Based on the most salient (about top 30-50) terms of each topic, it would be assigned (***annotated***) with a relevant emotion. For example: let's say there's a topic which has terms like 'father', 'mother, 'daughter', 'son'. It could be said that the context in this case is **'family'**, and perhaps the most relevant emotions for this topic would be **'love'** and/or **'trust'** (it would also depend on the other words in context).

Then, to verify how well the assignment of emotions has worked (within the constraints of the system), I would perform **Multi-label Classification**[5] on the then labeled dataset. In other words, I would take a portion of the dataset (the training set) and use it to train a few benchmark models and a final deeper neural network. The models would understand which words represent/express which emotions, and try to predict the same for the remaining portion of the dataset (the training set).

The Emotion Model



Robert Plutchik's Wheel of Emotions

Since I am in no way in a position to define what 'love' or 'trust' or any other arbitrary emotion is supposed to be, let's turn to the experts. The model of emotions that would be used in this project is **Robert Plutchik's Wheel of Emotions**[6] (pictured above). According to his Theory of Emotions, he considered there to be eight primary emotions: **anger, fear, sadness, disgust, surprise, anticipation, trust,** and **joy,** while emotions or *feelings* such as 'love', 'guilt', 'remorse', 'hope', etc, are combinations of these primary emotions.

Problem Statement

To reiterate, there are the problems:

- Topic Modeling
- Emotion Annotation
- Multilabel Classification

The **Topic Model** algorithm used in this project is **Latent Dirichlet Allocation**[7], with its implementation coming from the **SKLearn**[8] library, and **pyLDAvis**[9] is the library used for visualizing the topics. The major problem with topic modeling is finding the ***optimal number of topics*** that causes a proper division in the set of documents (documents in this case being songs). The word 'proper' is difficult to define in this context, as what's proper for me may not necessarily be proper for someone else, and I am not expert so that doesn't exactly help my cause. What I've tried to do in this project is find a division that 'looks' proper.

For example, let's say that there is one topic that represents anger and/or violence, with words such as 'gun', 'punch', 'shoot', 'kill', etc. or perhaps profanities, while there's another that represents love* with words such as 'love', 'heart', etc. or the words related to family mentioned earlier, another with words expressing 'sadness', while a few represent a mixture of the emotions. A division such as this one may be optimal.

*I would like to point out that if I refer to **love** as an emotion in this document, I am referring to: **love = joy + trust** (based on the Wheel of Emotions and Plutchik's theory.

I also believe it would be unlikely (though not impossible) for a song to express an emotion such as 'disgust', and a similar (but a bit more likely) case would be for 'surprise'. I would decide whether or not to consider these two emotions for classification based on how the output of the assignment turns out to be. This, again, being subjective, could be a conflict of interest, as someone may choose to disagree with me.

The **Emotion Annotation** is in itself a problem, and perhaps is more subjective than the creation of the topic model. Thankfully, I was able to find a dataset to help decrease this subjectivity. It would be described in the **Datasets** section of this report. The **Emotion Annotation** section would also discuss the problem of 'how' to use the dataset (as there are multiple options in terms of how to go through with using it), and whether or not to depend on it entirely. In short, it would be about choosing the threshold values for deciding which topics a song represents, and which emotions a topic represents.

After the dataset is labeled with emotions, the next step is **Multi-label Classification**. An example row perhaps would look like

artist	song	joy	trust	sadness	anger	fear	anticipation
eminem	mockingbird	1	1	1	0	0	0

This is assuming 'disgust' and 'surprise' don't factor in, they may. Even so, I think that for this particular song they should be off (0). So, each song would have each emotion either turned on or off, and the Multi-label Classification model, which would be a Neural Network created using **Keras** (with **TensorFlow** as the backend) , would try to learn and predict the values for each emotion, based on the lyrics, while trying to outperform the benchmark models while doing so.

Metrics

Based on the discussion above, it is likely that the dataset may have **label imbalance**. In other words, it's possible that an emotion such as 'joy' or 'trust' is more prevalent than 'anticipation' or 'surprise'. The metric used should be able to account for label imbalance, and that's why the **F-score**[10] **($F_{\beta=0.5}$)** is the evaluation metric for the multilabel-classification part of the problem. The other two problems are too subjective and hence are evaluated manually.

Datasets

The project uses the following datasets:

- Lyrics
- EmoLex

Using these two, a third dataset is created, which is then used for multilabel classification.

Lyrics Dataset

Getting the Dataset

The dataset is available at and can be downloaded from Kaggle here.

Data Exploration

The dataset (lyrics.csv) contains the **artist name, song title, genre, year,** and **lyrics** as columns. The lyrics are from MetroLyrics, and so most of them have probably been updated by users/visitors. 'lyrics' is the only column that is relevant to us for this project (the others wouldn't be dropped as they can help decide some context, but they wouldn't be used in any of the models themselves).

Here's a sample of the dataset:

artist	song	genre	year	lyrics
cocteau-twins	know-who-you-are-at-every-age	Pop	2006	It seems things are indicative to,\nA distinct...
bad-religion	the-world-won-t-stop	Rock	2006	You've got to quit your little charade\nAnd jo...
doris-day	t-ain-t-me	Jazz	2016	`Tain't me baby gee I'm blue\n`Tain't me that ...
beautiful-south	love-is	Pop	2006	Ooh you care, you really, really care\nFrom th...
george-strait	get-along-with-you	Country	2007	I don't get along\nWith people who aren't kind...

The lyrics are thus like regular documents, with characters such as the new-line character, and other punctuations present. We only need the **words** from the lyrics. Speaking of words, because of being updated by users, and also because of how words are used and pronounced in some genres of music, there are some words that are contractions (for example: **tryna** instead of **trying to**, **ova** instead of **over**, **neva** instead of **never**, etc) and some that have improper endings (such as: **livin'** instead of **living**, **tryin** instead of **trying**, etc.) These are things that need to be taken care of in **Data Preprocessing**.

There are some cases where the lyrics are not present. This 'missing data' (NaN) would be discarded. There are also cases where the lyrics are too small in length (sometimes just one word in case of instrumental music). Some artists have many songs, while some have few. There are also some non-English songs in the dataset.

Data Preprocessing

Before preprocessing, any song that does not have lyrics already is removed from the dataset. The preprocessing of is then handled within the text_utils.py file, mostly using regular expressions to 'clean' the text. It involves:

- Replacing any relevant unicode character with its ASCII equivalent (in this case: the comma),
- Fixing -ing endings (like lovin' with loving, etc)
- Using the Natural Language Toolkit's Part of Speech tagger to remove proper nouns
- Making the text lowercase, and removing punctuations
- Removing unnecessary contractions and fixing line-endings (as described above)
- Removing stop words (stop words are words that do not necessarily carry any contextual information). In this case, any 'emotional' information. The set of stop words was created and updated over multiple runs of the Topic Model - finding the most frequently occuring yet emotionally irrelevant words overtime. Some additional stop words were given by the Lexicon database (words that did not carry any of the primary emotions were considered stop words).

A new column called 'cleaned_lyrics' is added to the dataset, with only the relevant words and other characters.

The langdetect module is used to to **remove non-English songs**. Also, songs that did not contain **at least 100 words** after cleaning are also removed from the dataset.

As an example, this is what remains of Michael Jackson's Thriller:

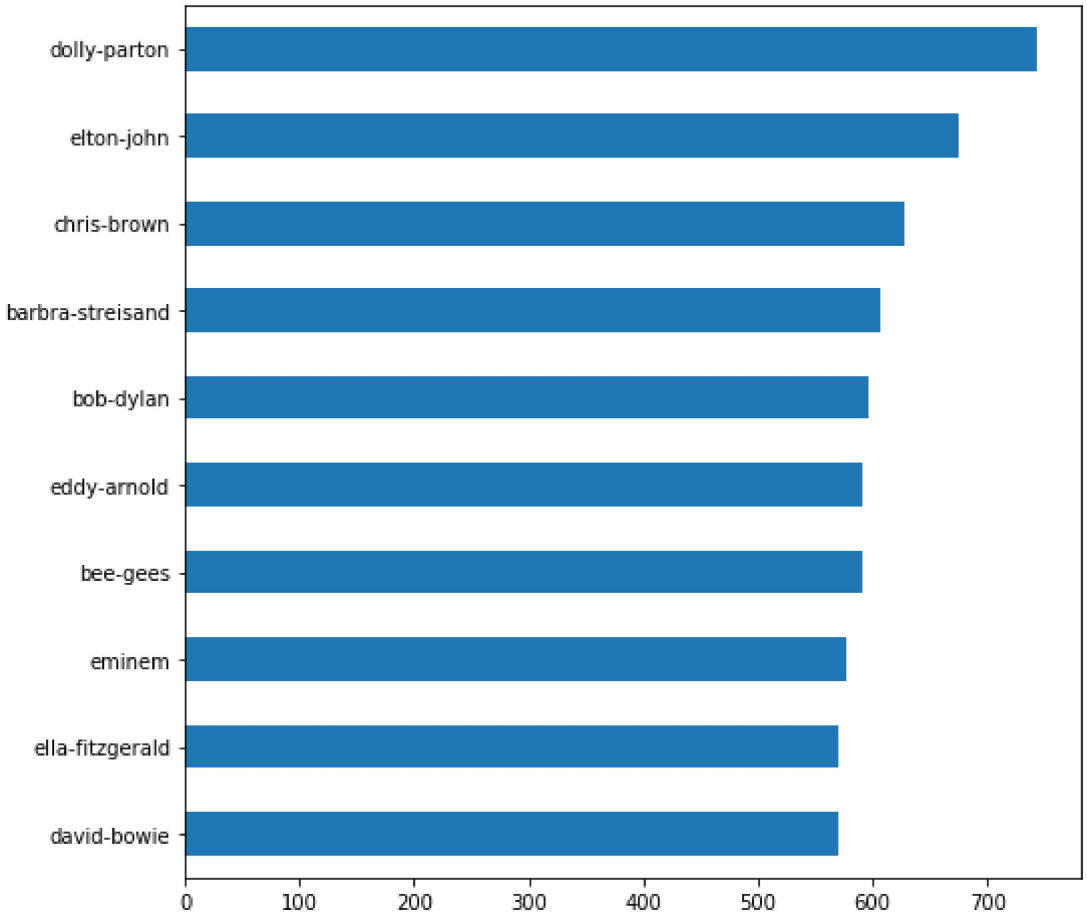
close midnight evil lurking dark moonlight sight stops heart scream terror takes sound start freeze horror eyes paralyzed thriller night save beast strike thriller night fighting life inside killer thriller tonight bring tonight hear door slam realize left feel cold hand wonder sun close eyes hope imagination hear creature creeping time thriller night save beast strike thriller night fighting life inside killer thriller tonight thrill tonight falls land midnight hour close hand crawl search blood terrorize neighborhood whosoever found soul stand face hounds hell rot inside

corpse shell thrill tonight thrill tonight babe thrill tonight thriller thriller thriller thriller thrill tonight thrill tonight thrill tonight thrill tonight thriller

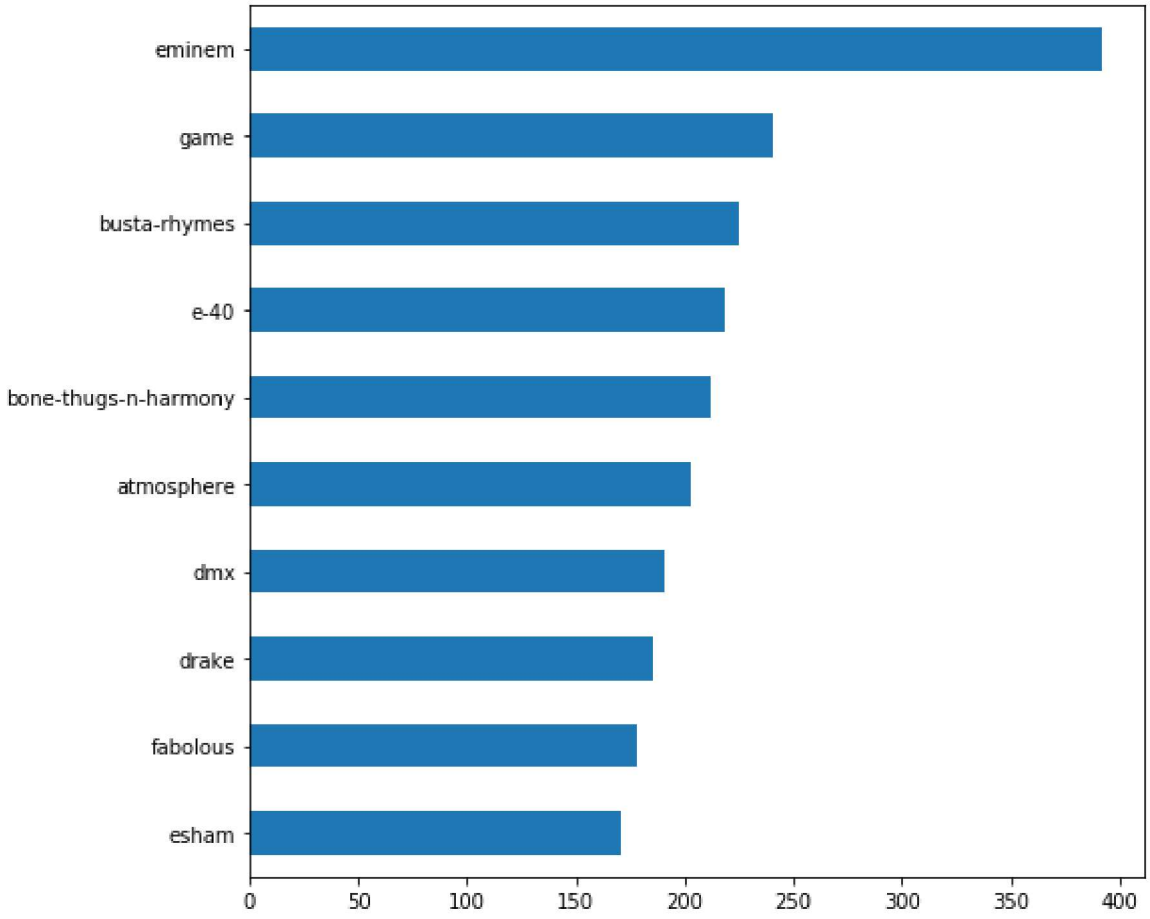
That's about 108 words after cleaning, reduced from the almost 300 words before. Sadly, no Michael Jackson songs are present in the dataset, but that's a problem for another day.

At this stage, the Lyrics dataset is ready to be vectorized and passed into the Topic Model. That would be discussed shortly.

For now, here's a look at the top ten artists (by number of songs) in the dataset **before** preprocessing:



And here's a look at the top ten artists **after** the preprocessing is over:



The reason for this change is that most of the artists in the latter graph are hip-hop artists (and rap songs contain more words and thus are usually longer). Most of the previous top artists' songs would have been dropped because the number of words remaining after preprocessing wouldn't have satisfied the minimum word requirement (100 words). There are still songs by artists such as Dolly Parton and Elton John, but they are not in hundreds as earlier. The monopoly of Eminem with the most number of songs is also discussed later in the Topic Model section.

EmoLex Dataset

Getting the Dataset

The dataset is not directly available for download. One needs to email the author of the dataset to receive access to it. More information on this page.

Data Exploration

The dataset is a compressed file containing multiple folders with text files. The text file of interest for this project is "**NRC-Emotion-Lexicon-Wordlevel-v0.92.txt**". It contains word-level primary emotion association for over 7000 words. About 3000 of those, do not carry any emotional context (and those words were added to the stop words to be removed from the Lyrics Dataset).

The text file is in the following format:

machine	anger	0
machine	anticipation	0
machine	disgust	0
machine	fear	0
machine	joy	0
machine	negative	0
machine	positive	0
machine	sadness	0
machine	surprise	0
machine	trust	1

Data Preprocessing

The file is parsed and changed into the following format:

word	anger	anticipation	disgust	fear	joy	sadness	surprise	trust	sum
conspiracy	0	0	0	**1**	0	0	0	0	**1**
inferno	**1**	0	0	**1**	0	0	0	0	**2**
bliss	0	0	0	0	**1**	0	0	0	**1**
regretted	0	0	0	0	0	**1**	0	0	**1**
bellows	**1**	0	0	0	0	0	0	0	**1**

The **'sum'** column contains the **sum** of all the emotions present for that word (as there may be multiple). It's created and used to eliminate the words that don't carry any emotions. The **'negative'** and **'positive'** sentiments from the text file are included in the parsed file but **not** included in the sum.

There are a few more changes made to this dataset, but that would be discussed in the **Emotion Annotation** section.

Vectorization and Embeddings

- CountVectorizer[11] vs TfidfVectorizer[12] (SKLearn)
- Word2Vec[13] (Gensim)

CountVectorizer vs TfidfVectorizer

The project was test run in its entirety (multiple times) with both **CountVectorizer** and **TfidfVectorizer**, individually as well as in tandem (CountVectorizer for Topic Modeling, TfidfVectorizer for Multi-label Classification, and vice versa).

Perhaps because many of the documents are very small in length, or because some words are more prevalent than others and Tfidf attempts to equalize them, it ended up making

the topics too indistinguishable, though using Tf-Idf for classification later on (with CountVectorizer for Topic Modeling) only slightly affected the F-score.

Either way, CountVectorizer was chosen because it offered superior performance in either case.

Word2Vec

The Word2Vec model was not initially in use, however it was Incorporated later on because of two reasons:

Words too many

As only the top 500 artists are considered for the dataset (the number was even lower earlier), the number of words in the vocabulary ended up being slightly greater than the number of samples (songs). Increasing the number of artists to increase the number of songs also caused an increase in the vocabulary size and also required me to adjust the parameters of the topic model (n_components). After doing this a few times I decided to limit the number of words all together. To so this, the cleaned lyrics are used to create a Word2Vec model, that drops those words that occur too few times. The vocabulary from this model is combined with the words that are present in the EmoLex database (with sum >= 1). Words from the cleaned lyrics that are not present in this combined set are dropped and the remaining words are added to a column called ‘constraint lyrics’. These constrained lyrics are then vectorized using the CountVectorizer.

Words too few

Quite ironically, the number of words from the set of top 50 words from each topic that were present in the EmoLex dataset was quite low. The Word2Vec embedding was used here to find words similar to the top words to increase the number of matches in the EmoLex dataset. Since the model is quite efficient at finding contextually similar words (for example: words similar to ‘father’, according to Word2Vec, can be ‘mother’, ‘daughter’, 'son', etc. while words similar to 'gun’ would be 'pistol’, 'shoot’, 'bullet’, etc.), it helped increase the number of matches and the chances of assignment.

Topic Modeling

Topic Modeling and Data Preprocessing was performed sort of simultaneously, especially in the early stages of the project. To determine an optimal number of topics, the topic model was created multiple times with different parameter values (and updated inputs), and many of these runs helped identify aberrant words - that were then classified as either stop words or contractions (or both). This also helped identify those artists whose lyrics were too convoluted for them to be included in the dataset, as too many stop words or contractions (expansions) would have been required for simplification.

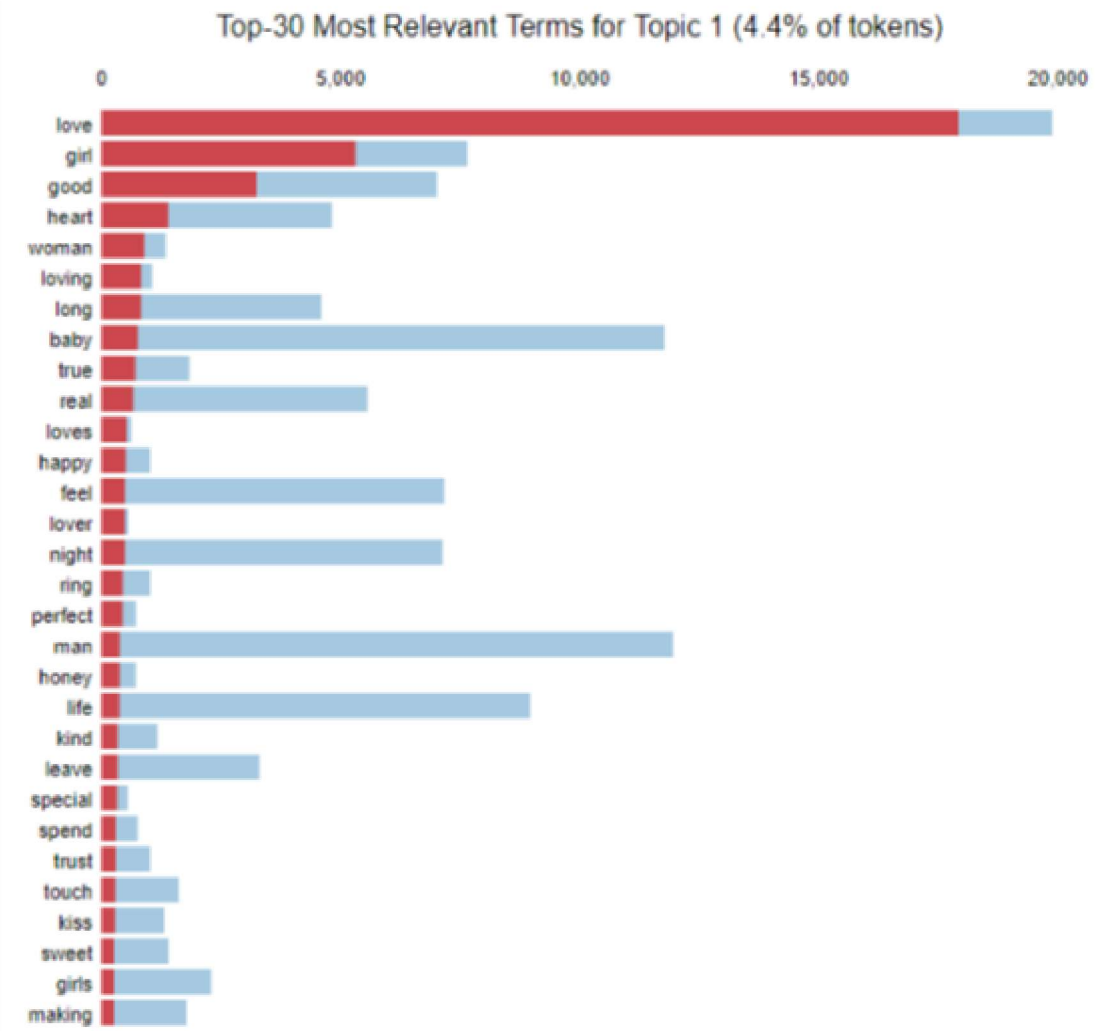
The values for number of topics that were tested were in the range [4, 32]. When the top 100 or 200 artists were being considered (earlier: during manual emotion assignment without EmoLex), the number of songs were somewhere between 6,000-8,000, the vocabulary size was upwards of 9,000 (pre-Word2Vec), and the number of topics that divided the data into sets of *seemingly readily identifiable* topics (in most cases) was 16.

After including EmoLex, when the number of artists to include was increased to 500, the number of topics had to increase to 18. At this point, the samples and the vocabulary size were both upwards of 11,000. The Word2Vec reduction caused the number of topics to be 20 and about halved the vocabulary size.

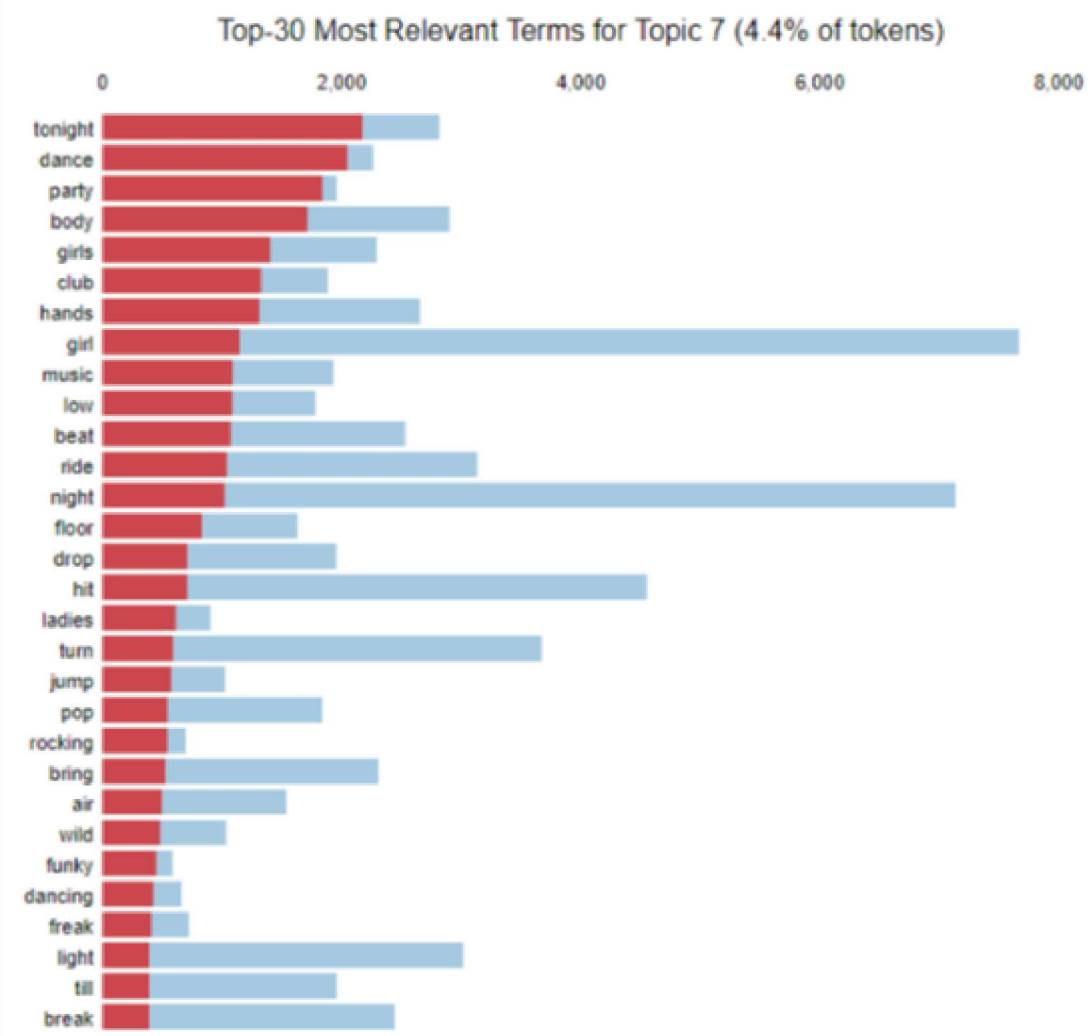
During this time, a function that assigns one or more topics to each song based on how relevant the topics are to that song was also created. This function is present in the **data_utils.py** file.

This process was difficult in the beginning, specially until the EmoLex Database was not in use. The difficulty was (and still is, though less) mainly in identifying emotions for some of the topics, specifically for cases where it wasn't as obvious for me, and to identify emotions other than love, anger and sadness. It became a bit easier after incorporating EmoLex. I was also able to understand how to identify an emotion such as anticipation after considering how it merges with other emotions (*anger* + *anticipation* = *aggression, fear* + *anticipation* = *anxiety*).

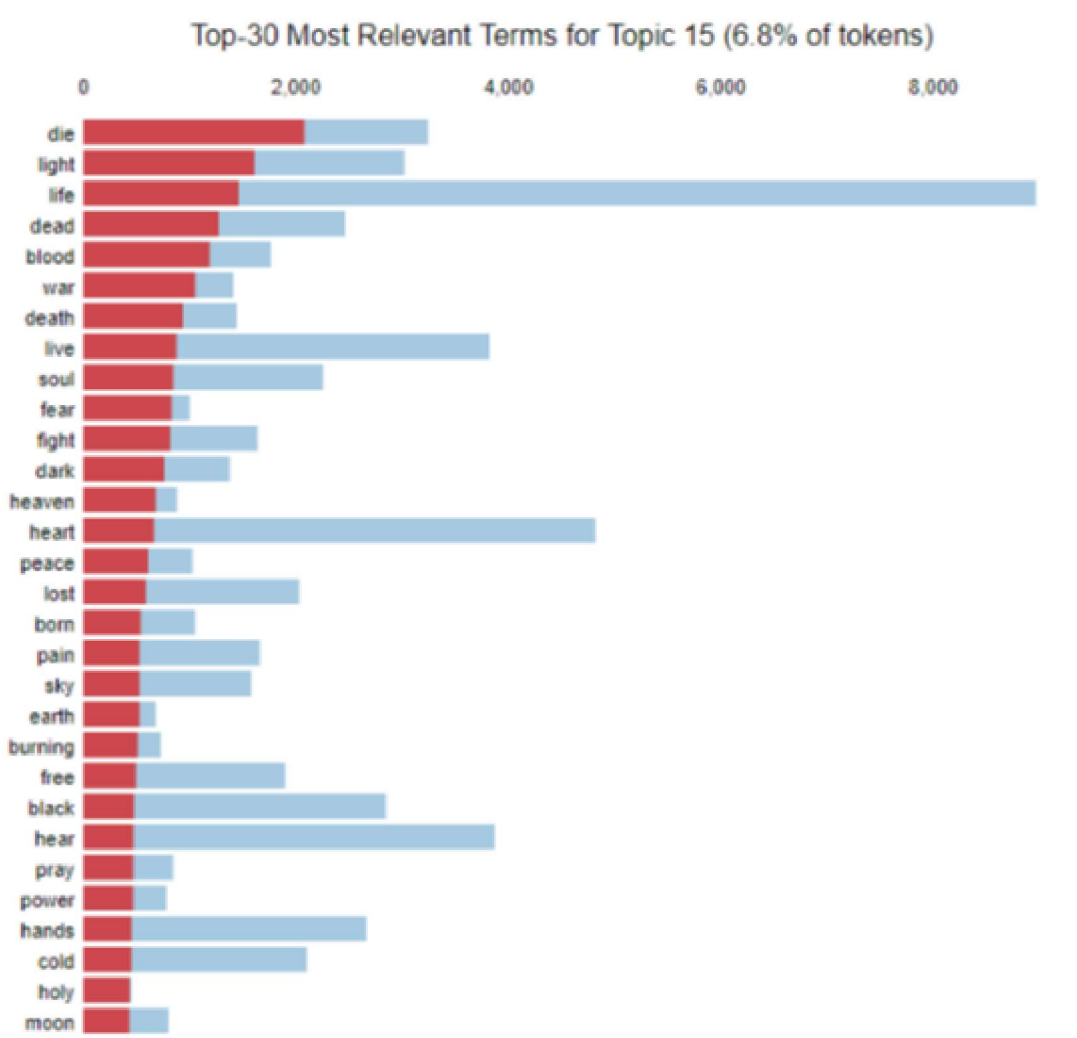
Below are some of the example top words, specially from the easily identifiable topics:



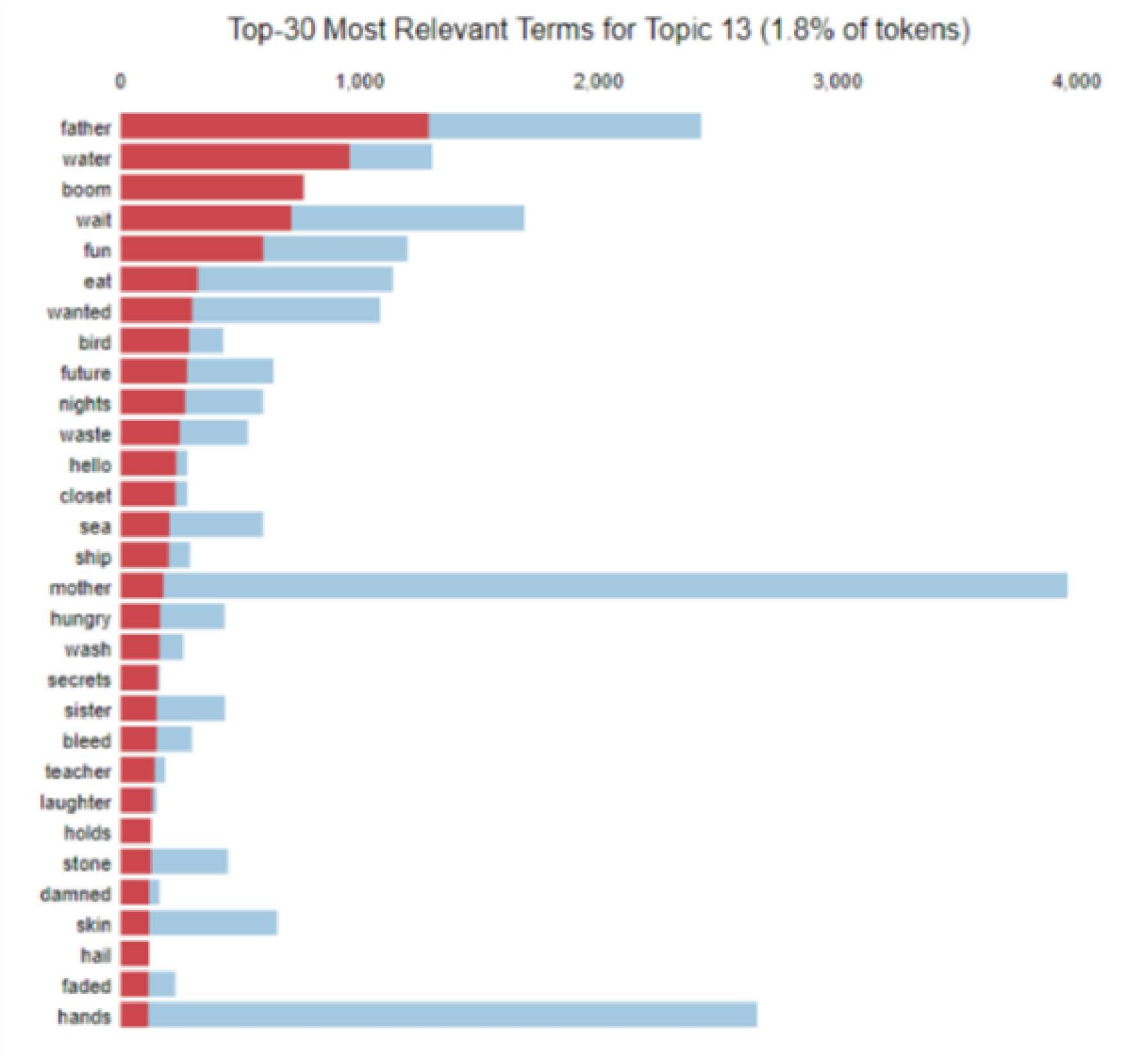
Top 30 Terms for a Topic representing ‘love’ (joy + trust)



Top 30 Terms for a Topic representing ‘joy’ (but not necessarily trust)



Top 30 Terms for a Topic representing ‘sadness’ (and perhaps anticipation?)



Top 30 Terms for a Topic representing ‘family’ in a way (could perhaps be assigned love?)

The image above also shows examples of how stop words such as ‘hands’ or ‘hello’ could be identified, though one may perhaps argue that ‘hello’ could be a gesture of trust.

Emotion Annotation

After the topic model is ready and songs are assigned their topics, the top words of each topic are used to assign emotions to the topics (which are then passed on to the songs). Identifying emotions for some topics is easy while not so much for others. However, with the EmoLex database, it becomes easier.

After receiving the preprocessed EmoLex dataset, a modification is made to it. The word is stemmed and stored in the 'stem' column of the dataset:

word	stem	anger	anticipation	disgust	fear	joy	sadness	surprise	trust	sum
abandon	abandon	0	0	0	1	0	1	0	0	2
abandoned	abandon	1	0	0	1	0	1	0	0	3
abandonment	abandon	1	0	0	1	0	1	1	0	4

The dataset is then grouped by the 'stem', and the emotions are added together.

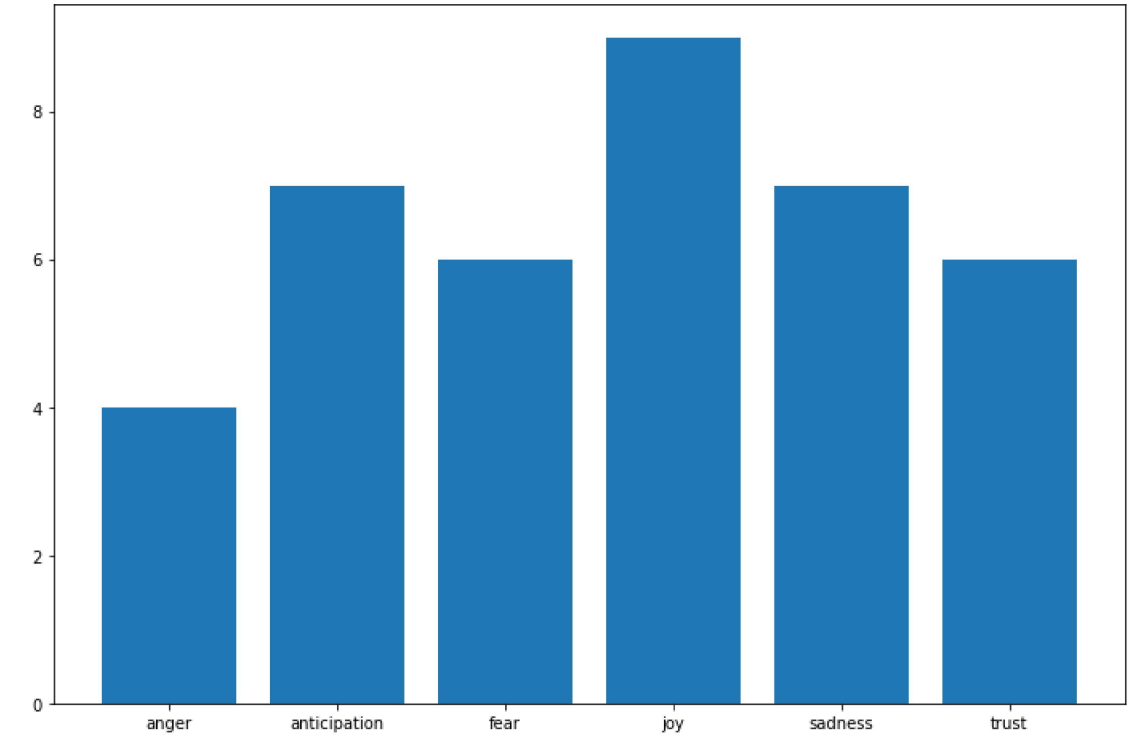
stem	anger	anticipation	disgust	fear	joy	sadness	surprise	trust
abandon	2	0	0	3	0	3	1	0

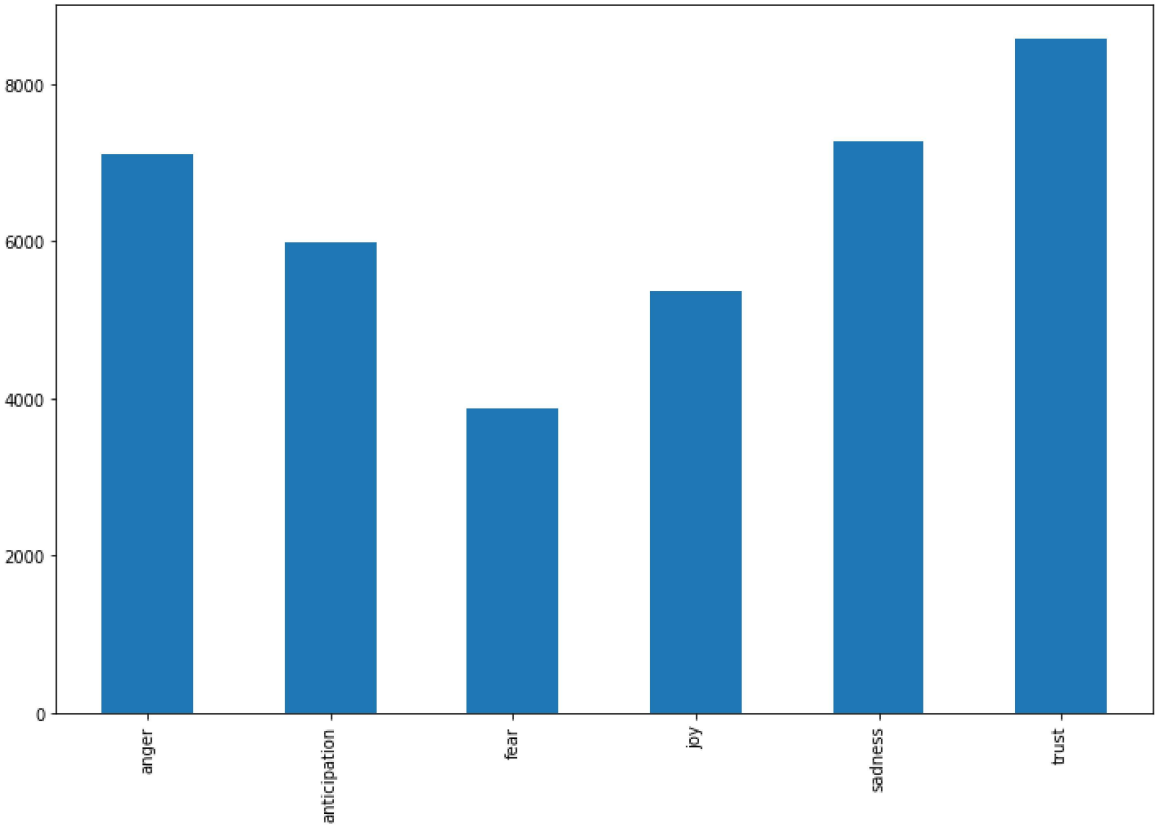
This is done to increase the number of matches using stemming (the words being matched are also stemmed before comparison).

Also, the words being matched are the top 50 words for each topic (and also the words *similar* to these top 50 words given by the Word2Vec model).

Manual adjustments are made to the emotions assigned to the topics where necessary, and then those emotions are forwarded to the songs (based on the topics which the songs represent).

Here's a look at a sample of how the emotions are distributed over topics and songs:


8
6
4
2
0
anger
anticipation
fear
joy
sadness
trust


8000
6000
4000
2000
0
anger
anticipation
fear
joy
sadness
trust

It should be noted that 'surprise' and 'disgust' were voluntarily dropped after not being too relevant (and also difficult to identify).

Multi-label Classification

After the dataset has been labeled, it is ready for classification. The input data are the same Count-Vectorized lyrics, and the output are the 6 emotions that were assigned. Tf-Idf and stemming were not used, as they were tried but didn't offer any improvements.

The neural network's input nodes are over 5000 in count, and the first layer reduces them to below a 1000 (to 256). The next layer expands its inputs to 512, and then finally shows the output. Many different combinations were tried before settling on to this one. There may be a better design possible, but I am not sure how to get there.

The final model was not made very large or deep, as doing so only increased the computation time, without improving (and sometimes decreasing) the F-score. The model is checkpointed at every epoch to calculate the F-score on the test-set and save the weights to a file if the F-score had improved.

For multilabel classification, the output layer used Sigmoid as the activation function, with Binary Cross-Entropy as the loss for the network (since the labels are on or off).

The model converges very slowly if Stochastic Gradient Descent is used as the optimizer, while Adam converges very quickly, and gets to highest F-score within 5 epochs.

Benchmark & Results

The project has three benchmark models for multilabel classification, against which the final model's efficiency (F-score) is compared:

- A **One Vs Rest Classifier**[14] using a **Support Vector Classifier**[15] with a *linear* kernel
- A **One Vs Rest Classifier** using a **Perceptron Classifier**[16]
- A **Multi-Layer Perceptron Classifier**[17]

All with their default parameters (except for the ones mentioned here and random_state for consistency across multiple runs).

The scores are as below:

Classifier	Training Time	Prediction Time	F-score
SVC	10m 13s 264ms	4m 51s 693ms	0.85
Perceptron	7s 791ms	440ms	0.85
MLPClassifier	2m 41s 481ms	150ms	0.87
Custom Model	38s 188ms	399ms	0.89

Reflection

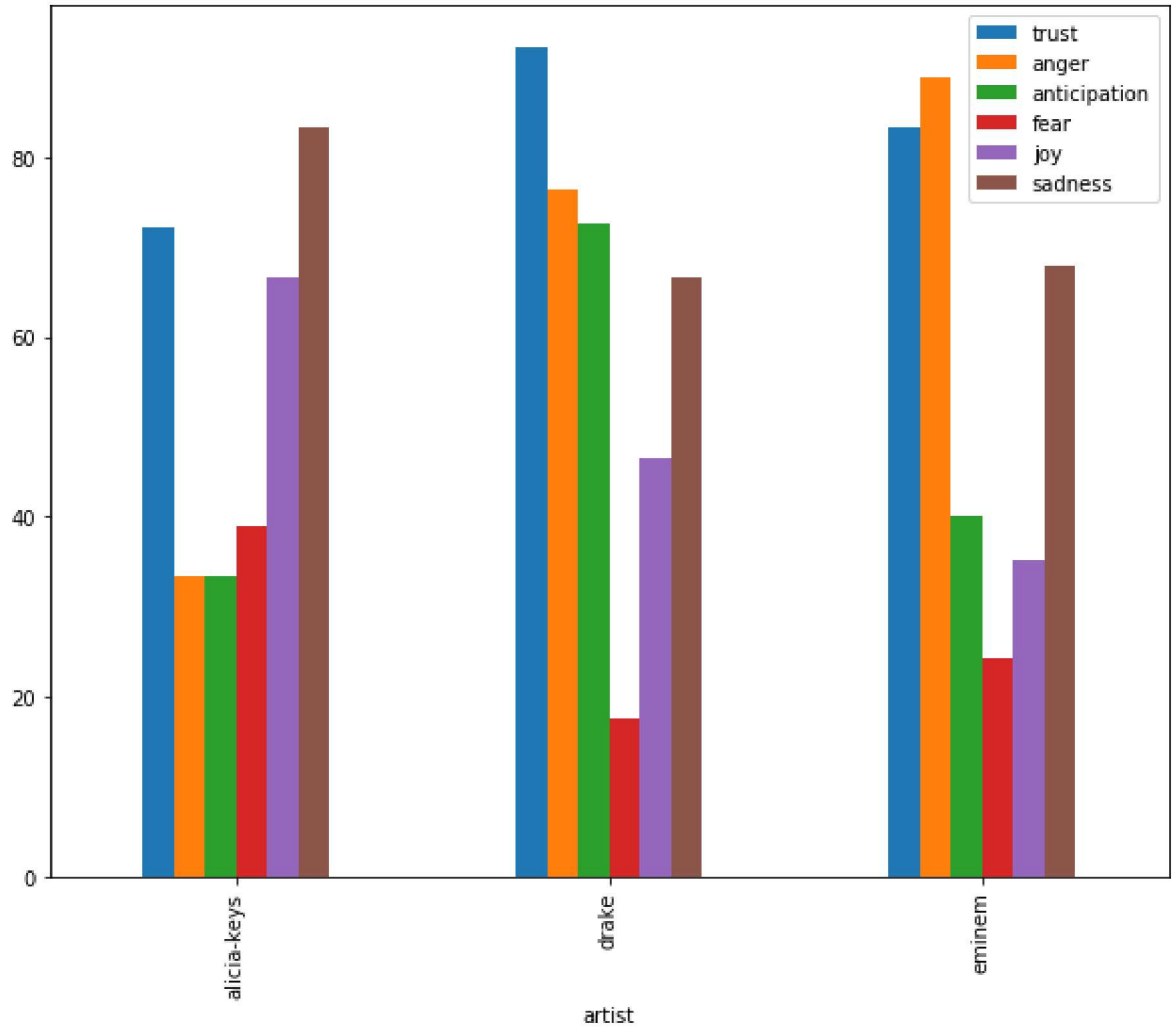
Overall, not a lot of improvement over the original models, perhaps it's because of how the dataset is labeled? Maybe a convolutional or a recurrent network would have gotten a higher score? I can't be sure.

A very surprising and quite honestly, pretty awesome thing that happened here is how the Perceptron model was able to be trained so quickly and give such a high score.

During the experiment, there was at one point a better result, with the MLP classifier being at 0.876, and the Custom Model making it upto 0.895 (in its current state, with the same number and sizes of layers). The difference was in the dataset as it had some words that had been removed later. I also feel like the dataset at this point was better tagged

There are still a lot of words in the dataset that are improperly formatted or spelled, it would be difficult and time consuming to find and fix them all (though it is possible, I believe).

After the dataset has been labeled this way, some interesting information can be extracted from it. For example:



That's the percentage of emotions present in songs of Eminem, Drake and Alicia Keys (based on this dataset). Eminem's music is usually echoes anger while Alicia Keys' is a bit sad. Trust has been very prevalent in the data, mostly due to EmoLex assignments.

Improvements

There are many ways to improve the results:

- My subjectiveness has been all over the project, if an expert (perhaps in Emotion Theory or Linguistics) were to look at this data and make changes, the results may be more appropriate.
- If instead of a simple neural network, a CNN or RNN is designed and applied, maybe it would be able to find better results?
- I believe using more data (more artists), and tweaking the system until the irrelevant words are identified and removed, and a bigger dataset is in use, the results would improve. During the course of the project, I gradually increased the number of songs, and the F-score and assignments seemed to get better as well.