Definition

Project Overview

Music Emotion Recognition is a concept in Music Information Retrieval which hasn't been around for a very long time. It is involved in determining or classifying emotions or moods that humans perceive in music using computer programs.

In this project, I focused on the use of Deep Learning, a field in Artificial Intelligence that applies a concept called Deep Neural Networks (DNNs) for learning representations or patterns from data, which helps the networks perform their specified tasks such as classification of labels. Originally DNNs were only used in the classification of images, but can now be used even on the classification of text as part of Natural Language Processing (NLP), more specifically Natural Language Understanding.

The project is built on the concepts of valence and arousal in music, which represent negative to positive moods, and calm to energetic moods respectively. The project decision, however, will be based on mood or emotion detection based on lyrics only as using audio will require higher data transfers and there may be copyright issues. The goal here is to use Deep Learning for the classification of music lyrics in text classification. For this project, I'll be applying a concept in the paper, "The emotions that we perceive in music: the influence of language and lyrics comprehension on agreement" by Juan Sebastián Gómez Cañón et. al, called 4Q, where emotional categories are further categorized into 4 Quadrant Dimensions of Arousal and Valence as shown in the table below;

Quadrant	**Emotions**	**Synonyms**
Q1 (A+V+)	Joyful activation	joy
	Power	-
	Surprise	-
Q2 (A+V-)	Anger	angry
	Fear	anguished
	Tension	tense
Q3 (A-V-)	Bitterness	bitter
	Sadness	sad
Q4 (A-V+)	Tenderness	gentle
	Peace	-
	Transcendence	spiritual

Therefore, the project will aim at classifying lyrics into either of the four quadrants in the range 0 to 3, i.e 0 == Q1 to 3 == Q4.

Problem Statement

Music recommendation has come a long way such that a single search of a song almost guarantees you a continuous stream of other songs that you might like, and almost always do.

These recommender systems usually use techniques such as collaborative filtering, and audio models to recommend music. This means that the music that gets recommended always has similarities with music you already listen to or what group of users (in collaborative filtering) you belong to.

Emotions in humans usually have a tendency to be erratic, and recommender systems sometimes won’t always be able to pick up on this since a user's current emotional state may force them to deviate from the usual user space or profile the system has built on them, meaning that a song being recommended may not be able to fit in well with a user’s current emotion, thus using music emotion recognition seems like something that will always be able to tailor music based on emotions, and be used by existing recommender systems to determine which songs based on the emotion they should recommend.

For most music streaming companies, the above-mentioned methods of music recommendation are mainly for financial gain, so they tend to be very effective in terms of getting music in front of the users of their platform, thus there is some margin of error in terms of the experience they are willing to forgo regardless of how it affects a user's experience. From my experience, their systems cannot usually create recommendations that necessarily have the emotional impact which as a listener, I may be in search of, especially when I need to slip into and out of certain moods.

Metrics

As for model evaluation, precision and recall were calculated. Precision refers to the number of correct labels among the labels predicted, while recall refers to the number of labels that were successfully predicted, among all the real labels. Accuracy was calculated although not used as the main measure of model performance since there is an existence of class imbalance in the dataset, thus doing having recall and precision will help in identifying the model’s performance on individual classes.

Analysis

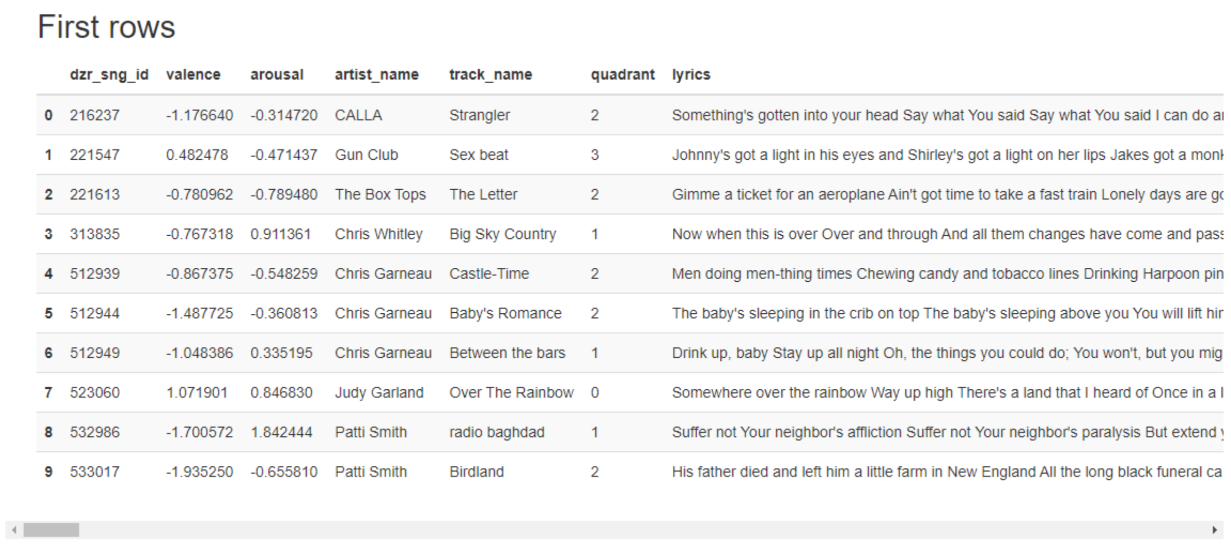
Data Exploration

The original dataset selected for this project was the Deezer Music Emotion Recognition (MER) Dataset which was mentioned in the paper “Music Mood Detection Based On Audio And Lyrics With Deep Neural Net” by Rémi Delbouys et. al, and was publicly available in the GitHub repository. The dataset came already split into training, validation, and testing sets, with data points 11000, 3000, and 3000 respectively. The dataset contained the following columns; dzr_sng_id, MSD_sng_id, MSD_track_id, valence, arousal, artist_name, track_name.

This dataset although was preprocessed to include every field but the MSD ones, and two new fields, quadrant, and lyrics were added. The quadrant was calculated using the quadrant table in the first section, whereas the lyrics were downloaded using the MusixMatch API. Since I could not hit the endpoint and directly add lyrics to our dataset, I build a lyrics_service.py program that could use query the endpoint using the track_name and artist_name and return the collected lyrics.

All data analysis-related tasks were performed using the [Pandas-profiling] library. Pandas profiling is an open-source Python module with which we can quickly do an **exploratory data analysis** with just a few lines of code. Besides, it also generates interactive reports in a web format that can be presented to any person, even if they don't know to program. The analysis here can be found in the EDA.html.

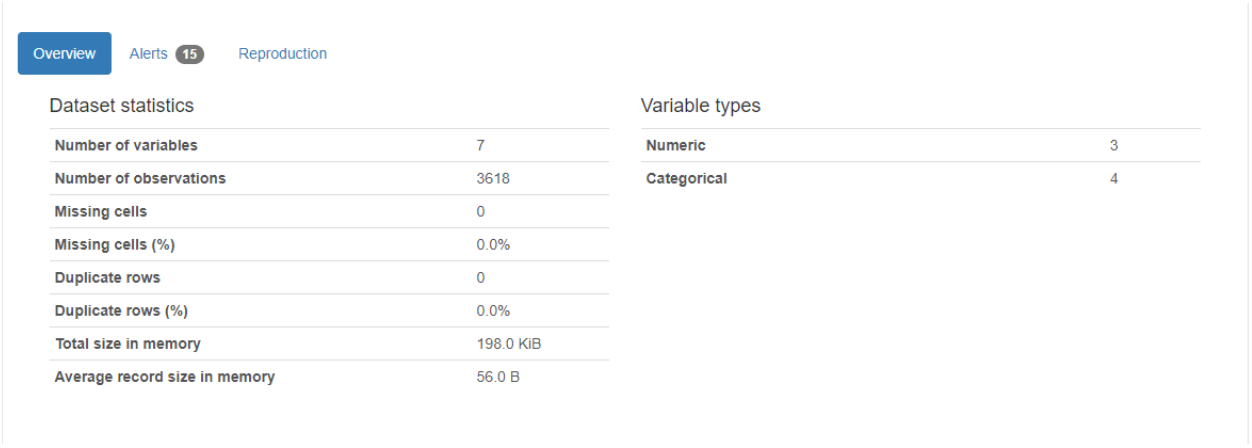
A sample of the data is as shown in the image below;



First rows

	dzr_sng_id	valence	arousal	artist_name	track_name	quadrant	lyrics
0	216237	-1.176640	-0.314720	CALLA	Strangler	2	Something's gotten into your head Say what You said Say what You said I can do a
1	221547	0.482478	-0.471437	Gun Club	Sex beat	3	Johnny's got a light in his eyes and Shirley's got a light on her lips Jakes got a mon
2	221613	-0.780962	-0.789480	The Box Tops	The Letter	2	Gimme a ticket for an aeroplane Ain't got time to take a fast train Lonely days are g
3	313835	-0.767318	0.911361	Chris Whitley	Big Sky Country	1	Now when this is over Over and through And all them changes have come and pas
4	512939	-0.867375	-0.548259	Chris Garneau	Castle-Time	2	Men doing men-thing times Chewing candy and tobacco lines Drinking Harpoon pin
5	512944	-1.487725	-0.360813	Chris Garneau	Baby's Romance	2	The baby's sleeping in the crib on top The baby's sleeping above you You will lift hi
6	512949	-1.048386	0.335195	Chris Garneau	Between the bars	1	Drink up, baby Stay up all night Oh, the things you could do; You won't, but you mig
7	523060	1.071901	0.846830	Judy Garland	Over The Rainbow	0	Somewhere over the rainbow Way up high There's a land that I heard of Once in a
8	532986	-1.700572	1.842444	Patti Smith	radio baghdad	1	Suffer not Your neighbor's affliction Suffer not Your neighbor's paralysis But extend
9	533017	-1.935250	-0.655810	Patti Smith	Birdland	2	His father died and left him a little farm in New England All the long black funeral ca

An overview of the data is shown in the image below. We get to see that there were no duplicates or missing data. The numeric types included dzr_sng_id, valence, arousal, and categorical were artist_name, track_name, quadrant, and lyrics fields.



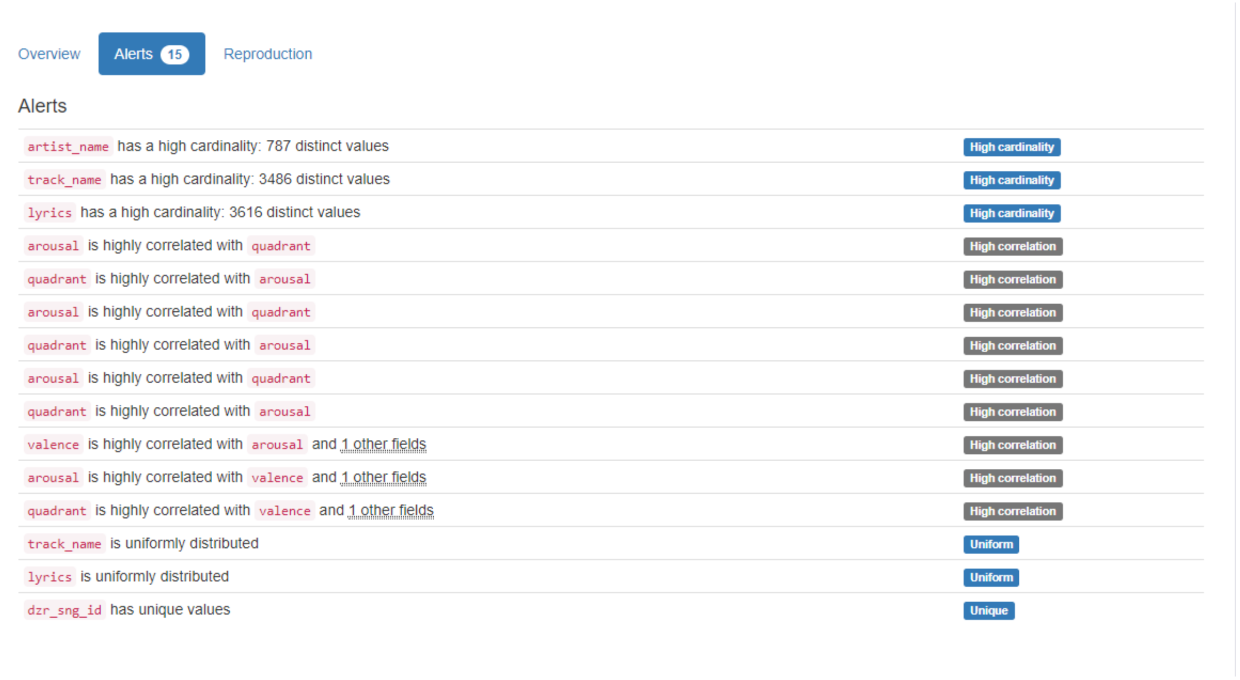
Dataset statistics

Number of variables	7
Number of observations	3618
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	198.0 KiB
Average record size in memory	56.0 B

Variable types

Numeric	3
Categorical	4

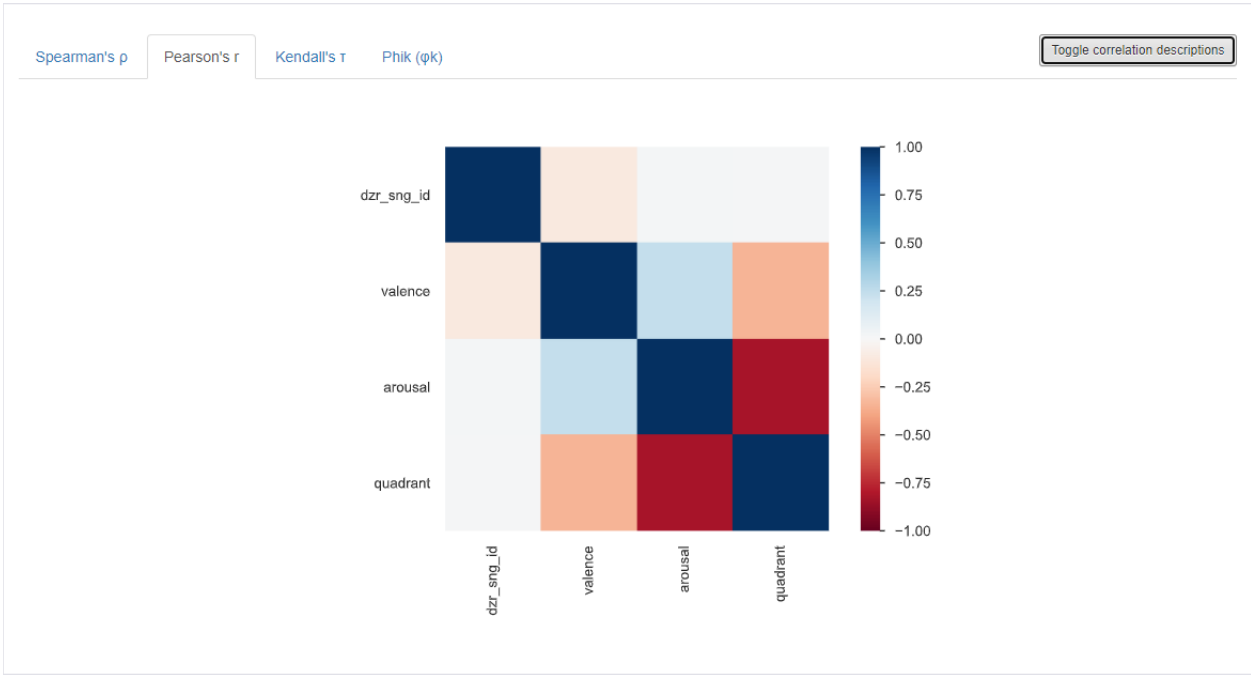
Some alerts raised by the profiler were as in the image below. Some fields such as artist_name, track_name, and lyrics had high cardinality, which was expected as having uncommon datapoint values for these was rather expected. It is also notable to mention the high correlation between arousal and valence, to quadrant. This shows that using the two to train the model would yield much as the quadrant field was a direct derivation of the two.



Alerts

`artist_name` has a high cardinality: 787 distinct values	High cardinality
`track_name` has a high cardinality: 3486 distinct values	High cardinality
`lyrics` has a high cardinality: 3616 distinct values	High cardinality
`arousal` is highly correlated with `quadrant`	High correlation
`quadrant` is highly correlated with `arousal`	High correlation
`arousal` is highly correlated with `quadrant`	High correlation
`quadrant` is highly correlated with `arousal`	High correlation
`arousal` is highly correlated with `quadrant`	High correlation
`quadrant` is highly correlated with `arousal`	High correlation
`valence` is highly correlated with `arousal` and 1 other fields	High correlation
`arousal` is highly correlated with `valence` and 1 other fields	High correlation
`quadrant` is highly correlated with `valence` and 1 other fields	High correlation
`track_name` is uniformly distributed	Uniform
`lyrics` is uniformly distributed	Uniform
`dzr_sng_id` has unique values	Unique

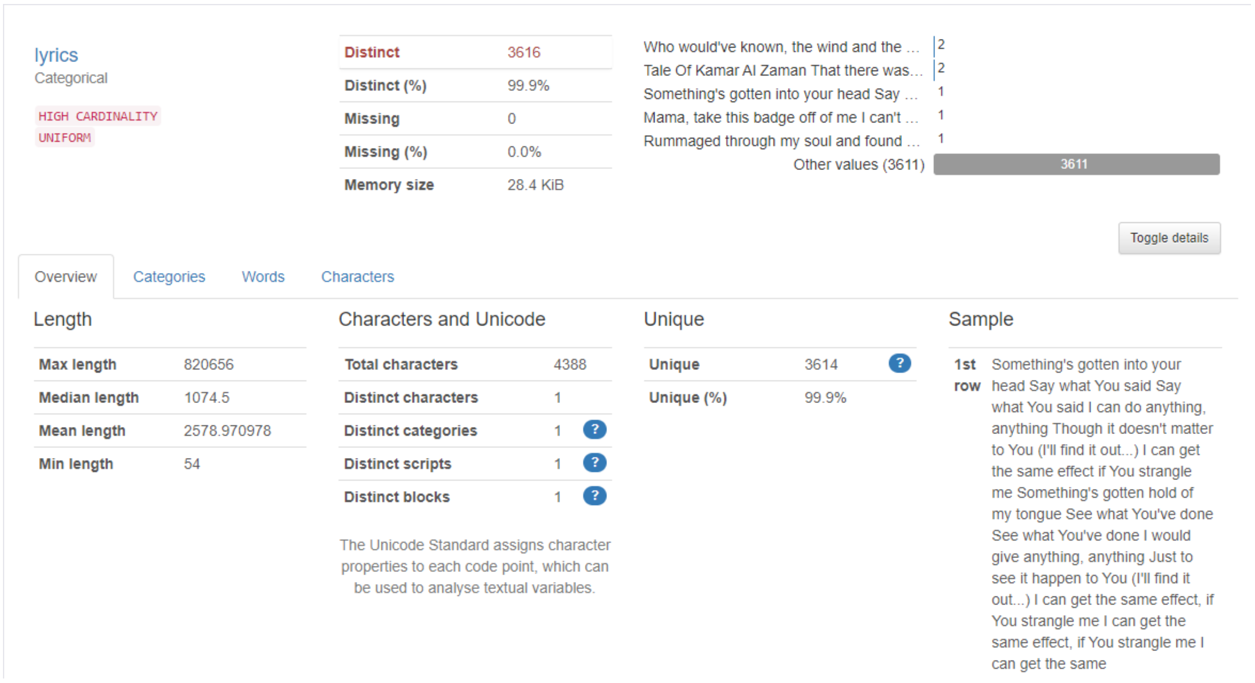
Pearson’s r correlation matrix was also created as shown in the image below. Its value lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation.



Exploratory Visualization

In this section, I focused more on the fields that were relevant to achieving the goal of the project. This included the lyrics field and the quadrant field, which is the label.

A simple overview of the lyrics field is as shown in the image below


lyrics
Categorical

HIGH CARDINALITY
UNIFORM

Distinct	3616
Distinct (%)	99.9%
Missing	0
Missing (%)	0.0%
Memory size	28.4 KiB

Who would've known, the wind and the …	2
Tale Of Kamar Al Zaman That there was…	2
Something's gotten into your head Say …	1
Mama, take this badge off of me I can't …	1
Rummaged through my soul and found …	1
Other values (3611)	3611

Length

Max length	820656
Median length	1074.5
Mean length	2578.970978
Min length	54

Characters and Unicode

Total characters	4388
Distinct characters	1
Distinct categories	1
Distinct scripts	1
Distinct blocks	1

The Unicode Standard assigns character properties to each code point, which can be used to analyse textual variables.

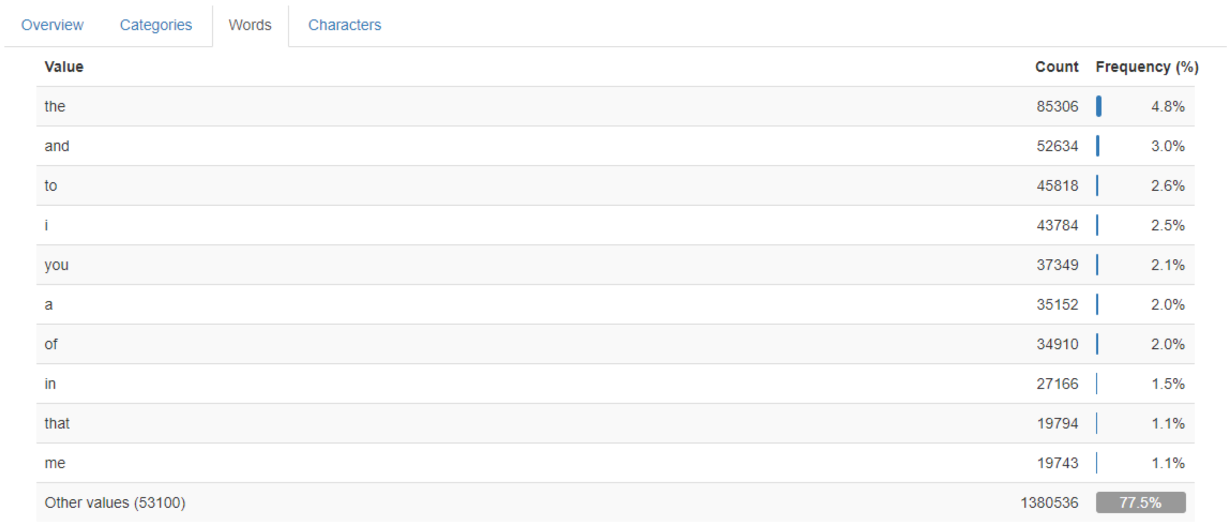
Unique

Unique	3614
Unique (%)	99.9%

Sample

1st row	Something's gotten into your head Say what You said Say what You said I can do anything, anything Though it doesn't matter to You (I'll find it out...) I can get the same effect if You strangle me Something's gotten hold of my tongue See what You've done See what You've done I would give anything, anything Just to see it happen to You (I'll find it out...) I can get the same effect, if You strangle me I can get the same effect, if You strangle me I can get the same

Through this step, I was also able to identify some of the most common words, as shown below. The article “the” was the most recurring word with a count of 85306 which is about 4.8% of the words in the data. Words like these will be removed during preprocessing as they may not carry any sentiment to them.

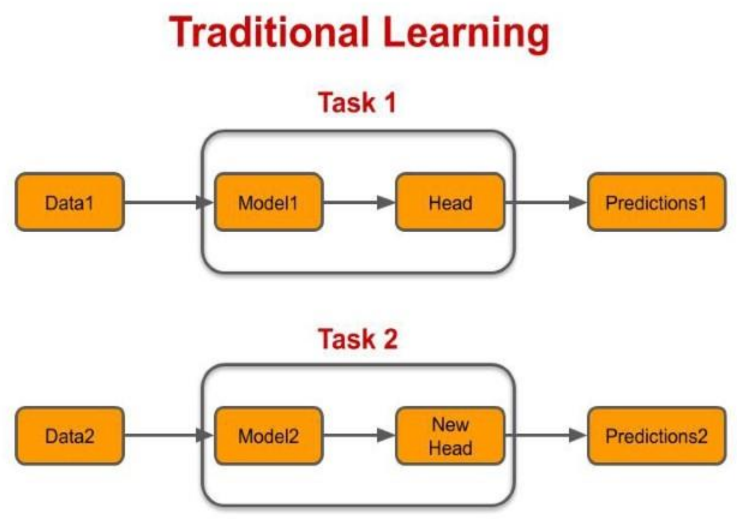

Overview
Categories
Words
Characters

Value	Count	Frequency (%)
the	85306	4.8%
and	52634	3.0%
to	45818	2.6%
i	43784	2.5%
you	37349	2.1%
a	35152	2.0%
of	34910	2.0%
in	27166	1.5%
that	19794	1.1%
me	19743	1.1%
Other values (53100)	1380536	77.5%

Algorithms and Techniques

The algorithm that I used was a pre-trained BERT model. This model was selected as it is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering, text classification, and language inference, without substantial task-specific architecture modifications, as from the paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” by Jacob Devlin et. al.

The technique that was used for training, therefore, was transfer learning. This is a research problem in Machine Learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. It can be considered the reuse of a pre-trained Deep Learning (DL) or ML on a new problem, in this case, a BERT model trained on unlabeled text to the classification of lyrics. A simple illustration is as shown in the figure below;



Benchmark

The benchmark projects for this project were more related to emotion recognition using audio rather than text, although they still gave a picture as to the kind of performance I should be expecting. Most of the information used for this section was gotten from this site, paperswithcode.com. The paper that contained information related to emotion recognition in music was “Codified audio language modeling learns useful representations for music information retrieval” by Rodrigo Castellon et. al, which demonstrated the usage of a technique called CALM with a performance of about 72.1% on arousal, and 61.7% on valence, both described in the first section. This averages to about 69.9%.

There were no benchmark models publicly available for using lyrics only for this particular project, although, since the dataset and the paper “Music Mood Detection Based On Audio And Lyrics With Deep Neural Net” by Rémi Delbouys et. al, were both from researchers at Deezer, I can safely assume that their platform uses some aspect of MER for the recommendation of music, and the paper utilizing CALM gave a somewhat related accuracy that I should expect.

Methodology

Data Preprocessing

Since the data being used is text, the following data preprocessing steps were taken;

- Lowering text. This is because I wanted to have words such as “happy” and “Happy” being tokenized to the same value.
- Removal of contractions. These are words that have been shorted by dropping one or more letters, e.g “aren’t” to “are not”.

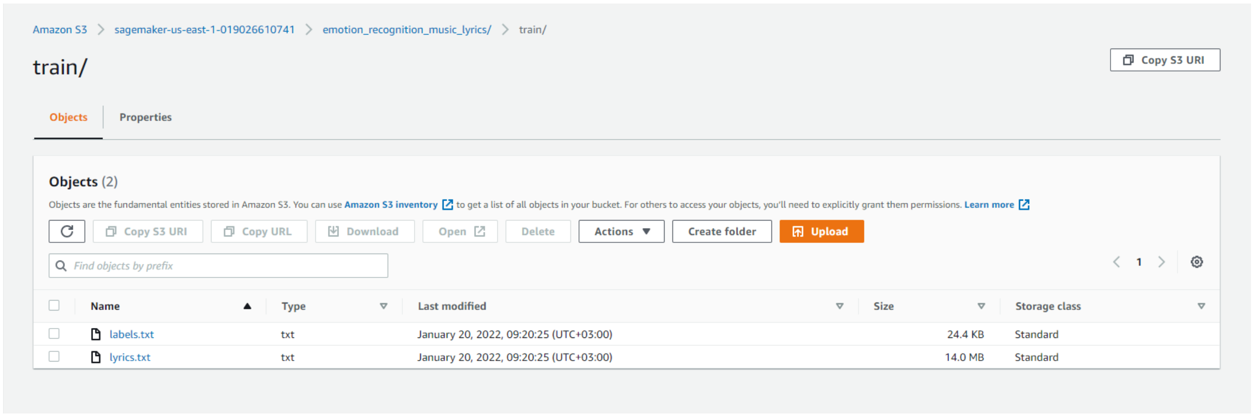
- Removal of punctuation marks.
- Removal of stopwords. These are commonly used words in a language e.g “the”, “a” etc.
- The last step was to remove any lyrics with a size less than or equal to 64. This helped ensure that I didn’t try to tokenize an empty string after the first four preprocessing steps.

All these were done using a combination of the re, and nltk python packages. The last step was achieved using pandas as the data was originally in a .csv format. A sample of before and after preprocessing is as shown below;

BEFORE: “I let the beast in too soon I don't know how to live without my hand on his throat I fight him always and still Oh darling, it's so sweet You think you know how crazy How crazy I am You say you don't spook easy You won't go But I know And I pray that you will Fast as you can, baby Run, free yourself of me Fast as you can I may be soft in your palm But I'll soon grow hungry for a fight And I will not let you win My pretty mouth will frame the phrases…”

AFTER: “let beast soon know live without hand throat fight always still oh darling sweet think know crazy crazy say spook easy go know pray fast baby run free fast may soft palm soon grow hungry fight let win pretty mouth frame phrases disprove faith man catch trying find way heart skin fast baby scratch free fast fast baby scratch free fast sometimes mind shake shift time…”

All the data was then moved into text files which were uploaded to an S3 bucket in AWS as shown in the image below;



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

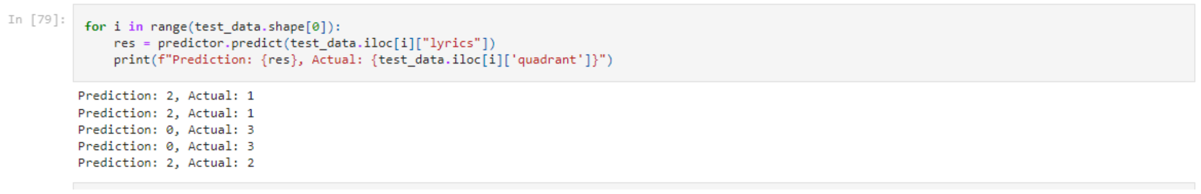
Name	Type	Last modified	Size	Storage class
labels.txt	txt	January 20, 2022, 09:20:25 (UTC+03:00)	24.4 KB	Standard
lyrics.txt	txt	January 20, 2022, 09:20:25 (UTC+03:00)	14.0 MB	Standard

Implementation

For this section I followed a four-step procedure as described below;

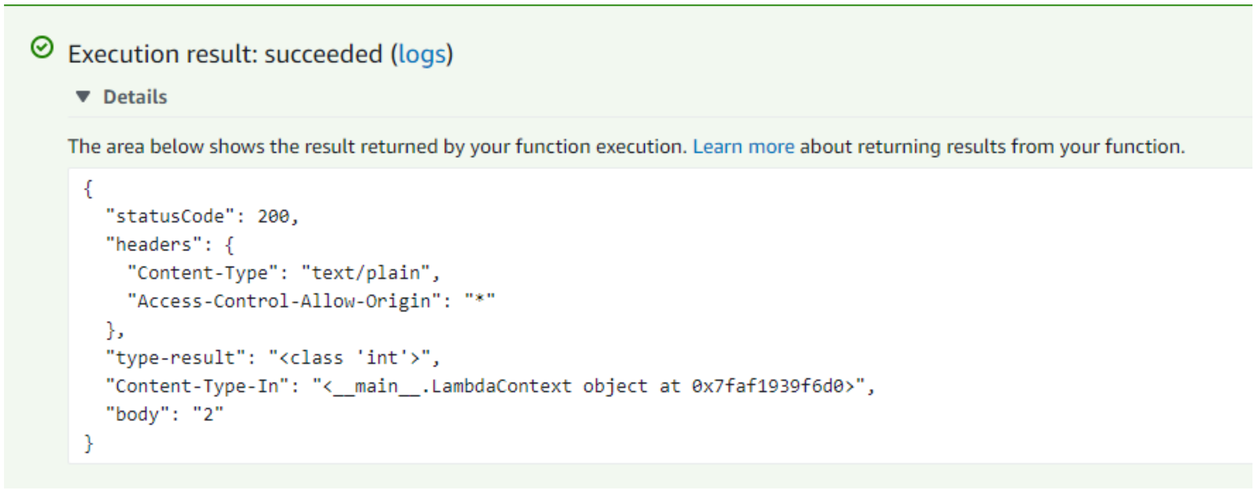
1. Hyperparameter Tuning. The purpose of this step was to perform a hyperparameter tuning job, where I trained 4 different models then selected the best performing one. To achieve this, I first created a hpo.py file that contained code to define the network, data reading, and loading, training, testing and saving the model. The objective metric goal, in this case, was to minimize the validation loss.

2. Training Job. After obtaining the best hyperparameters for the initial step, I created and fit a HuggingFace estimator using a PyTorch backend. I also added some profiler and debugger configs to the estimator and set up hooks in the train.py file.
3. Endpoint Deployment. This step involved deploying the trained model from step 2. Since deploying it from the same estimator fails to load the model as I had not included deployment-specific code in the train.py file, I decided to use the HuggingFaceModel class for deployment. I also created an inference.py file which contained three main functions, model_fn for model weights loading, input_fn for inputs processing, and predict_fn for making a prediction using the loaded model. Documentation for these can be found through this link. Sample output from invoking the endpoint from SageMaker is as shown in the image below:



```
In [79]:  for i in range(test_data.shape[0]):
              res = predictor.predict(test_data.iloc[i]["lyrics"])
              print(f"Prediction: {res}, Actual: {test_data.iloc[i]['quadrant']}")

Prediction: 2, Actual: 1
Prediction: 2, Actual: 1
Prediction: 0, Actual: 3
Prediction: 0, Actual: 3
Prediction: 2, Actual: 2
```

4. Lambda Function. In this step, I created a Lambda function to query or perform inference on the deployed endpoint. The initial design for this was to include two Lambda functions in a step function, one for preprocessing the lyrics and the other for performing inference. This architecture changed since the preprocessing happens inside the deployed endpoint, unforeseen in the initial project design. The function takes in an event containing the lyrics, invokes the endpoint then returns a JSON Object with status 200 OK and the prediction as shown in the images below;

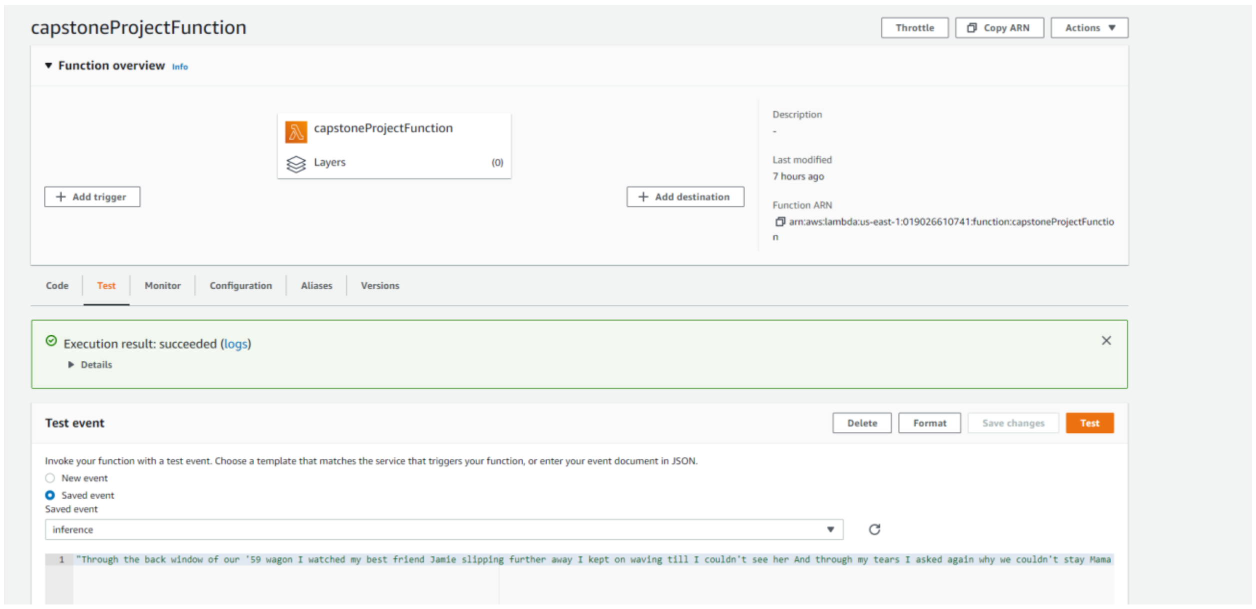


Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'int'>",
  "Content-Type-In": "<__main__.LambdaContext object at 0x7faf1939f6d0>",
  "body": "2"
}
```



Description
-
Last modified
7 hours ago
Function ARN
arn:aws:lambda:us-east-1:019026610741:function:capstoneProjectFunction

Execution result: succeeded (logs)

Invoke your function with a test event. Choose a template that matches the service that triggers your function, or enter your event document in JSON.

New event
[✓ Saved event]

Saved event
inference

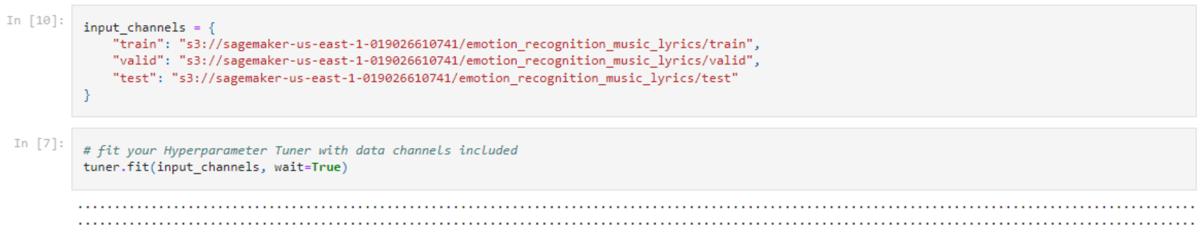
```
"Through the back window of our '59 wagon I watched my best friend Jamie slipping further away I kept on waving till I couldn't see her And through my tears I asked again why we couldn't stay Mama
```

Challenges faced here included the following;

- FileNotFoundError: [Errno 2] No such file or directory: '/opt/ml/input/data/training/train/lyrics.txt'. The solution to this was creating SM_CHANNEL variables for the datasets as shown below. This ensured that the code was searching the correct directory during data loading.

```
parser.add_argument("--train-dir", type=str, default=os.environ["SM_CHANNEL_TRAIN"])
parser.add_argument("--valid-dir", type=str, default=os.environ["SM_CHANNEL_VALID"])
parser.add_argument("--test-dir", type=str, default=os.environ["SM_CHANNEL_TEST"])
```

  Fitting was then done as shown below:



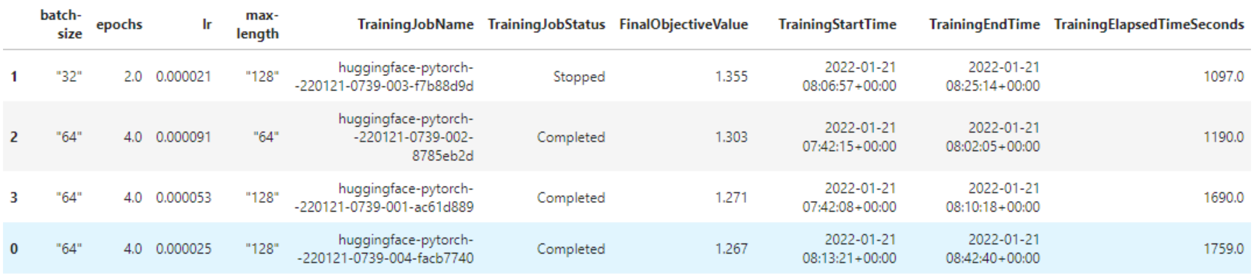
```
In [10]: input_channels = {
             "train": "s3://sagemaker-us-east-1-019026610741/emotion_recognition_music_lyrics/train",
             "valid": "s3://sagemaker-us-east-1-019026610741/emotion_recognition_music_lyrics/valid",
             "test": "s3://sagemaker-us-east-1-019026610741/emotion_recognition_music_lyrics/test"
         }

In [7]:  # fit your Hyperparameter Tuner with data channels included
         tuner.fit(input_channels, wait=True)

         ...........................................................................................................................
         ...........................................................................................................................
```

- Kernel dying during training job. The solution for this was to attach back to the running training job using sagemaker.estimator.Estimator.attach(jobname).
- TypeError: Object of type 'Tensor' is not JSON serializable. Occurred during endpoint deployment. This was the most challenging one as the Traceback error was not very clear, and following documentation from SageMaker still led to the same error. After

more than 10 deployments, I realize that I needed to convert the final model output in my inference.py file to a list.

Refinement

For refinement, I opted to use the Hyperparameter Tuning step as it enabled me to run multiple model training jobs concurrently. A table of the training jobs, showing their performance on the objective metric and the hyperparameters used is as shown in the image below:



	batch-size	epochs	lr	max-length	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
1	"32"	2.0	0.000021	"128"	huggingface-pytorch--220121-0739-003-f7b88d9d	Stopped	1.355	2022-01-21 08:06:57+00:00	2022-01-21 08:25:14+00:00	1097.0
2	"64"	4.0	0.000091	"64"	huggingface-pytorch--220121-0739-002-8785eb2d	Completed	1.303	2022-01-21 07:42:15+00:00	2022-01-21 08:02:05+00:00	1190.0
3	"64"	4.0	0.000053	"128"	huggingface-pytorch--220121-0739-001-ac61d889	Completed	1.271	2022-01-21 07:42:08+00:00	2022-01-21 08:10:18+00:00	1690.0
0	"64"	4.0	0.000025	"128"	huggingface-pytorch--220121-0739-004-facb7740	Completed	1.267	2022-01-21 08:13:21+00:00	2022-01-21 08:42:40+00:00	1759.0

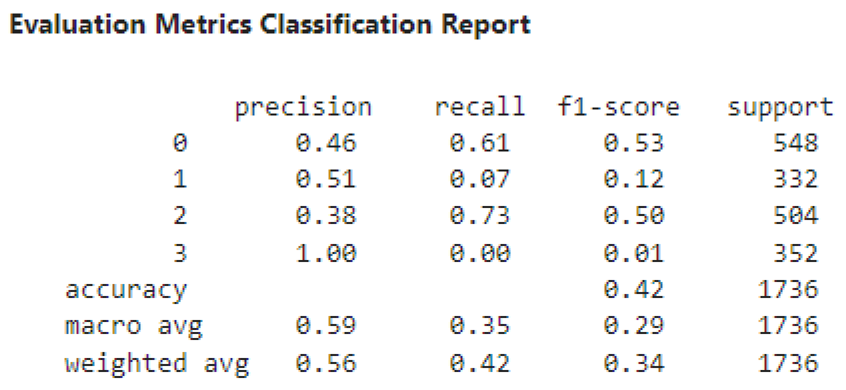
The last model performed the best with the following hyperparameters {'batch-size': '64', 'max-length': '128','epochs': '4', 'lr': '2.4834388581766214e-05'}, which were used to train the deployed model.

Results

Model Evaluation and Validation

Training of this model utilized the use of training data and validation data, this helped ensure that model testing only used data that the model had not seen before. Performance of the model on validation data during training can be seen in the table above, in the Final Objective Value field.

As for evaluation on the test set, the metrics used have been discussed in the metrics section in the first chapter. The classification report generated was as shown below:



Evaluation Metrics Classification Report

```
              precision    recall  f1-score   support
           0       0.46      0.61      0.53       548
           1       0.51      0.07      0.12       332
           2       0.38      0.73      0.50       504
           3       1.00      0.00      0.01       352
    accuracy                           0.42      1736
   macro avg       0.59      0.35      0.29      1736
weighted avg       0.56      0.42      0.34      1736
```

The model’s overall accuracy was about 42%, although per class performance on precision and recall going up to 61% and 73% on classes 0 and 2 which was expected as they represented Happy and Sad respectively, which were very high in the dataset.

Justification

Most users have a tendency of searching for music that is in sync with their current mood or emotional state, and since music streaming platforms usually recommends music based on past listens and a listener’s similarity with others, they usually miss to meet these states such that a listener gets presented with a mix-match of what they might like which may deviate from their current interests emotionally.

This project’s justification is tied in well into using Deep Learning to aid in music classification or clustering and the creation of playlists or clusters that the streaming platforms can tap into to recommend music based on emotion.

Thresholding the model’s performance to the benchmark would in some sense show that the model didn’t significantly enough solve the problem as near benchmark performance was not reached consistently. This performance shows that the model may have the potential to improve given more data or even combining it with audio data.