CSUN

# Lyrics Sentiment Analysis NLP Project

COMP541

Mrunal Prakash Gavali, Jefferson Perez Diaz

## INTRODUCTION

The goal of this project is to predict the overall net positive and negative sentiments aroused by songs using two supervised learning algorithms: Long short-term memory (LSTM) and Random Forest Generation Classifier using the Spotify Valence attribute as the label for binary classification. The Spotify Valence is a number between 0 and 1 reflecting the measure of musical positiveness conveyed by a track, where the valence score closer to 0 depicts negativity and closer to 1 depicts positivity. Based on the spotify website reference [4], the way they assessed this feature is through lyrics and audio mood measures. Both our models have 73% accuracy after training the models with 15k balanced data derived from the original larger 150k Kaggle Spotify lyrics dataset due to computational constraints.

## 2. RELATED WORK

Nowadays, many researchers working on sentiment analysis choose to apply machine learning and deep learning technologies. However, Seo and Jun-Ho. [2] seek to address this topic by a study on 40 people where they asked the group to classify 100 different styles of songs into 4 emotions: calm, sad, excited and happy. Their results showed that the average rate of correctly predicted songs from this test was 74% ranging from 70% to a maximum of 79%. They picked those dependent features through the use of stepwise regression procedure and analyzed the songs by taking into consideration 4 features of a song: Average height, Peak average and BPM as they noticed, there was an increment of accuracy when combining these features.

Chen and Li [3] using convolutional neural networks-long short-term memory (CNN-LSTM), also agreed with Seo and Jun-ho's work. In the study, the researchers trained the model with a million songs, using audio features and also words for prediction. After evaluating different techniques, while using different features of music, they found that CNN-LSTM performed the best with an average accuracy of 78% when both audio features and lexical features were taken into account [3].Work by Agarwal et al. [1] is also worth mentioning as he contributed to this topic by using only lyrics using modern NLP techniques like context based transformer models. On the other hand, in this project motivated by aforementioned research we seek to predict positive or negative feelings aroused by songs using the spotify valence attribute for supervised learning and sentiment prediction.

## 3. DATA CLEANING

Initially the data set consisted of more than 150,000 rows. Each row represents one song in the data and it included five columns: unnamed column (denoted the row number), artist, seq, song, and label. To work with this data, we changed the names of the variables to ones that better represented those features, and dropped the unnamed column from the data. The data now contained 4 columns: artist, lyric, title and rating. To remove ambiguity, we decided to only consider the lyrics with extreme positive label (greater than or equal to 0.67) and extreme negative label (lesser than or equal to 0.33) for training our model. This removed about 52,000 songs, leaving the dataset with 98,000 songs in it. Additionally, we also removed non-english songs from the dataset. Also, we decided to expand the contractions of the words so that we could later remove pronouns as it does not impact the overall net sentiment of a song. Once that was done, we converted every word to lowercase to ensure no same word would get counted multiple times, removed numbers, removed punctuations and finally, lemmatized the words and also removed songs that had very few words in it as it was not going to add much meaning to the data. Next, we decided to remove swear words, stopwords like a, of, to , at, etc., and open lyrical syllables like mm, ooh, la, pre, woo, etc., that we would encounter in the dataset to reduce the amount of words in the song as they do not affect the net overall sentiment of songs. For all those words, we made a vector that included it and decided to remove it from the "lyric" column. After applying all of these techniques to reduce the unnecessary chunks of words in the data and implementing dimensionality reduction of textual data, we proceeded to pick a perfectly balanced amount of positive and negative songs for training our model. We agreed to select 7500 song lyrics for each positive and negative class, that is, a total of 15000 song lyrics of balanced data containing equal amount of positive and negative songs.The decision was done to maintain the computational time of our algorithms low, while at the same time having enough data to get a good performance on both models. With all these steps mentioned above, the final new cleaned data ended up with 15000 rows, having the same amount of positive and negative songs in it. An example of the clean data can be seen in figure (1). By observing the new cleaned data, we can tell how our strings got reduced to a much smaller form without losing the words that are important for the overall net sentiment of the songs. With the above mentioned, the data is ready to be vectorized so that our data mining techniques can understand the strings.

**Clean dataset**

table shows a few lyrics after having been cleaned

| artist | lyric | title | rating |
|---|---|---|---|
| Freedom Call | child bear forever young alone virging s hand son mankinds innocence raise holy land alone star shine thousand night diamond sky freedom us high sky guide path life save us despair blaze ray light feel warm inside take us holy land day come return innocence pass gate time guide light saviour soul keep us alive eye open wide stare crystal ball time come us return innocence forever forever whenever whenever shine starlight saviour come somewhere ride guardian take us back eagle dare fly fantasy take us eternity never cry never die return innocence forever forever whenever whenever shine starlight saviour come somewhere ride guardian take us back eagle dare shine starlight saviour come everywhere ride guardian take us back eagle dare tonight child bear forever young fly bring us back wayward son cry freedom us shine starlight savior come somewhere ride guardian take us back eagle dare | Shine On | 0 |
| Skye Sweetnam | see boyfriend hang girl hate tell last night late can believe tell lie come undo now live run look numb one take ride city get place just can stand pity tear fall face use think just begin now live run look numb one one day see dream one day say one hitch ride greyhound stash away train buy ticket subway play guitar rain want follow river island sun now live run look numb one day see dream one day say one late night start fight just may think right now feel fine good without one day say one one day see dream one day say one know one day see haunt dream one day say one numb one | Number One | 1 |
| Everything But the Girl | still can believe two now just friend get lover free weekend know say first meet must always play hand get two year lose streak get back together get back together something can next week get back together get back together get back try get back together match make heaven make right earth search starry sky end arm teach know just think can another arthur marilyn now two year lose streak get back together get back together something can next week get back together get back together get back try get back together say get bore can say get bore people search whole life never find show still can believe two just friend get lover loose end girl listen say let boy just walk away wo get back try get back together | Get Back Together | 1 |
| Freddie & the Dreamers | little truth every lie tell little good bad doyesss instead run away listen say want just little want say word love little love gonna make dream come true instead run away listen say want just little ooh nothing world can say love true baby want little day nighttime just little little truth every little lie tell little good bad yes instead run away listen say want just little ohhhh child say want just little | A Little You [*] | 1 |

*Figure (1): Cleaned Data Sample*

Table 1: Most Common Terms (Cleaned data)

| Word | Frequency |
|---|---|
| can | 29388 |
| get | 28112 |
| love | 26807 |
| know | 22728 |
| like | 17766 |
| just | 16795 |
| come | 15279 |
| good | 14800 |
| say | 14782 |
| now | 14012 |

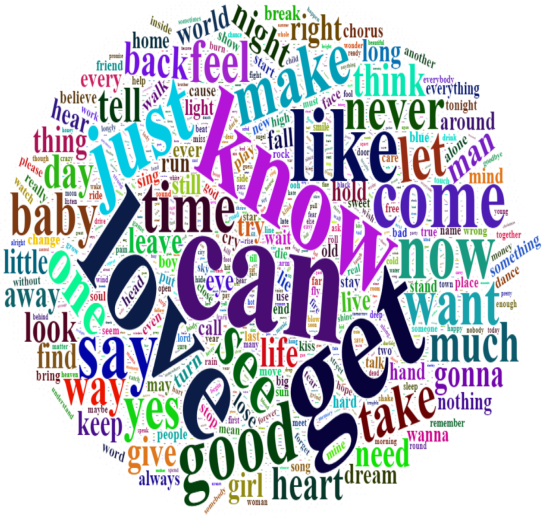



*Figure (2): Word frequency table*

*Figure (3): Wordcloud*

Other statistics of the data are shown in the figure (2) above. It is very noticeable that by cleaning the data we manage to not only reduce the amount of noisy data, but also increase the frequency of some words in our dataset. Both figures (2) and (3) suggest that the word “can” is the most common term in the data followed by “get” “love”, and “know”. Finally, the Figure (4)

below shows a comparison of the words that are most frequently used in negative and positive songs.

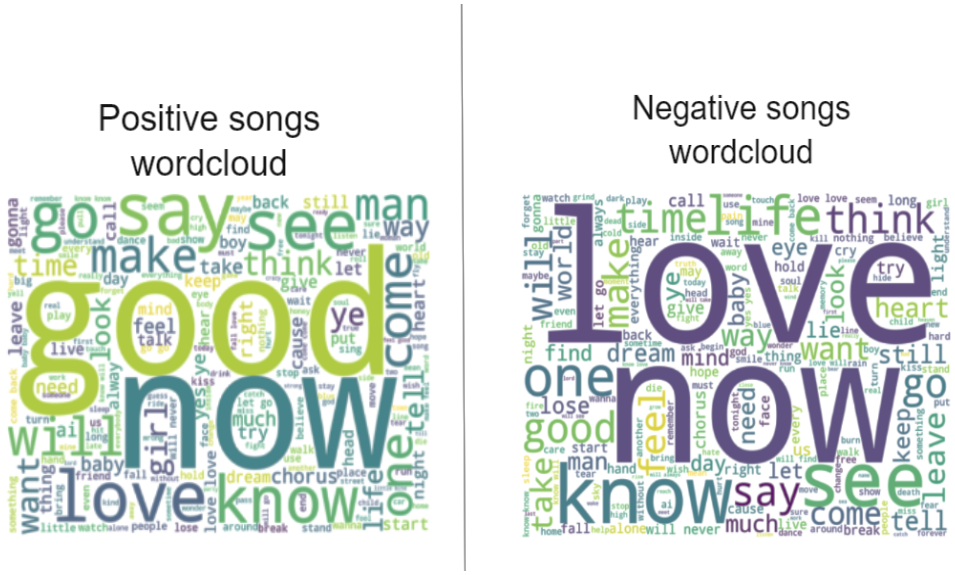



*Figure (4): Word Cloud of Positive and Negative songs*

This image entails that some of the words that could potentially induce positive sentiments are also part of the negative songs. This may be due to many aspects, one of them being, the whole context of the sentences. When you combine the whole context of a sentence then some words might be classified differently as the overall meaning of the sentence gets shifted after combining words that seemed to be positive with words that induce negative sentiments. Another reason why this could be possible, might have to do with the way Spotify labels the valence for a song. Our assumption is that the spotify valence was calculated by taking into consideration the audio features of a song based on the Audio API reference by spotify [4].

## 4. VECTORIZATION TECHNIQUE FOR RANDOM FOREST

After the data has been cleaned, to make the Random Forest Generation classifier understand the strings, the data needs to be vectorized first. Vectorization is the process of transforming our data into numeric values. Initially, I decided to use a bag of words dataset which is a vectorizer method that works by counting the amount of times a word repeats in the string, to then create columns for each word encountered in the whole data and add the total amount of times that word repeated in the specific row which is the representation of where that word belongs to. In the process, I removed words that did not appear frequently to reduce the dimensionality of the data. This allowed the vectorizer to remove names and other words that were simply not adding or impacting the overall sentiment of the song. The final outcome is the following:

**Vectorization technique**

table shows the first 20 songs with a few selected words

| able | abuse | ache | acs | act | action | add | admit | adore | advice | affair | afraid | afternoon | age | ago | rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Figure (5): Vectorization technique (Bag-of-words) for Random forest classification*

The vectorization technique created 1428 new columns/features including the dependent feature and with each row representing the song after removing the sparsity terms, that is, the terms that do not appear frequently in the dataset. While this method performs fairly decent with classifiers, it does a poor job at selecting the features to keep in the dataset, as it only looks for the frequency of an specific feature in the whole dataset, (we set the removal of the infrequent feature, if the feature did not appear at least 0.0049% in the entire data) without taking into account the weight that a word can have in the song, There is a even better approach to solve this issue and that is using the tf_idf vectorizer.

Term frequency - inverse document frequency (known as TF-IDF) solves the issue of removing words that can potentially shift a song's overall net sentiment. The idea of TF-IDF is similar to the bag of words but in this case it also takes into account terms that are rare but are still important to the overall outcome of the prediction for a song.TF-IDF vectorizes the strings in a better way and even normalizes the features giving the following result. TF-IDF created a

**Vectorization technique**

table shows the first 20 songs with a few selected words

| can | get | love | know | go | like | just | come | good | say | now | see | make | time | one | rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.00000000 | 0.00000000 | 0.05691372 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0 |
| 0.00000000 | 0.00000000 | 0.04525476 | 0.00000000 | 0.22077518 | 0.09655410 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.10203124 | 0.00000000 | 0 |
| 0.01696319 | 0.00000000 | 0.00000000 | 0.11254469 | 0.00000000 | 0.00000000 | 0.08189193 | 0.13180787 | 0.00000000 | 0.06575392 | 0.06775148 | 0.02145913 | 0.00000000 | 0.04576140 | 0.02360517 | 0 |
| 0.00000000 | 0.00000000 | 0.05947110 | 0.00000000 | 0.05802590 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0 |
| 0.01245082 | 0.02953621 | 0.01489775 | 0.01376779 | 0.00000000 | 0.07946342 | 0.06010788 | 0.00000000 | 0.01666792 | 0.03217516 | 0.36467883 | 0.01575080 | 0.01664229 | 0.00000000 | 0.06930386 | 1 |
| 0.15605162 | 0.00000000 | 0.00000000 | 0.00000000 | 0.22772839 | 0.00000000 | 0.04708492 | 0.00000000 | 0.10445323 | 0.00000000 | 0.00000000 | 0.00000000 | 0.15643890 | 0.00000000 | 0.00000000 | 0 |
| 0.23366054 | 0.15653153 | 0.09023183 | 0.02084698 | 0.03301467 | 0.07219340 | 0.05688404 | 0.00000000 | 0.02523832 | 0.02435958 | 0.03764941 | 0.01192481 | 0.00000000 | 0.11443297 | 0.02623472 | 1 |
| 0.00000000 | 0.33452423 | 0.10123830 | 0.00000000 | 0.00000000 | 0.00000000 | 0.37442679 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.03567839 | 0.00000000 | 0.00000000 | 0.07849287 | 0 |
| 0.12906691 | 0.07654412 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.12535993 | 0.00000000 | 0.00000000 | 0.04295809 | 0.04081874 | 0.04312908 | 0.13056837 | 0.04490085 | 0 |
| 0.08834197 | 0.00000000 | 0.07046911 | 0.06512415 | 0.00000000 | 0.00000000 | 0.14216065 | 0.00000000 | 0.15768465 | 0.03804860 | 0.03920449 | 0.11175623 | 0.03936054 | 0.00000000 | 0.20488748 | 0 |
| 0.01289562 | 0.00000000 | 0.03085995 | 0.00000000 | 0.01505501 | 0.01646045 | 0.00000000 | 0.05010096 | 0.00000000 | 0.00000000 | 0.01716850 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0 |
| 0.16492694 | 0.00000000 | 0.00000000 | 0.03039531 | 0.09627211 | 0.00000000 | 0.00000000 | 0.07119556 | 0.03679798 | 0.21310051 | 0.03659573 | 0.17386614 | 0.00000000 | 0.00000000 | 0.00000000 | 1 |
| 0.04213885 | 0.04998153 | 0.10084064 | 0.04659603 | 0.02459753 | 0.05378758 | 0.02542882 | 0.02728571 | 0.00000000 | 0.10889440 | 0.02805063 | 0.05330738 | 0.02816228 | 0.00000000 | 0.00000000 | 1 |
| 0.02108844 | 0.05002664 | 0.00000000 | 0.02331904 | 0.00000000 | 0.00000000 | 0.00000000 | 0.32772401 | 0.02823112 | 0.00000000 | 0.02807595 | 0.02667774 | 0.00000000 | 0.00000000 | 0.00000000 | 1 |
| 0.00000000 | 0.05347960 | 0.10789818 | 0.09971430 | 0.26319041 | 0.23020807 | 0.00000000 | 0.00000000 | 0.06035938 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.06026656 | 0.12163344 | 0.00000000 | 0 |
| 0.04813577 | 0.00000000 | 0.40317067 | 0.37259089 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0 |
| 0.04494966 | 0.00000000 | 0.13445885 | 0.07455623 | 0.00000000 | 0.00000000 | 0.05425003 | 0.00000000 | 0.06017416 | 0.02903951 | 0.05984343 | 0.02843159 | 0.06008162 | 0.00000000 | 0.40657379 | 1 |
| 0.00000000 | 0.03667172 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.12510866 | 0.04302332 | 1 |
| 0.03078193 | 0.00000000 | 0.03683145 | 0.03403785 | 0.00000000 | 0.00000000 | 0.03715090 | 0.00000000 | 0.04120780 | 0.00000000 | 0.00000000 | 0.00000000 | 0.04114443 | 0.04152003 | 0.17133867 | 0 |
| 0.27416829 | 0.02501502 | 0.00000000 | 0.04664124 | 0.04924280 | 0.10767955 | 0.00000000 | 0.00000000 | 0.05646606 | 0.05450003 | 0.00000000 | 0.00000000 | 0.02818961 | 0.05689390 | 0.00000000 | 0 |

*Figure (6): TF-IDF vectorization for Random forest classification*

total of 1439 columns, where the last column is the dependent variable and each row represents the song where that word belongs to. Also, we applied dimensionality reduction by removing words that did not appear frequently( 0.0049%) after applying the formula. This method performs great with our data as it does a good job at keeping values that appear frequently in a row/song but not in the entire dataset. After this data transformation, the data is now ready to be fed into the random forest classifier and for making predictions.

## 5. METHODS

The data was perfectly balanced with 7500 positive songs and 7500 negative songs. Next, we split this 15000 rows of data to an 80-20 split, and proceeded to train the models with the training set which includes 12,000 songs in it and used the 3000 songs in the test set to evaluate the model performance.

### 5. 1 DATA MINING TECHNIQUE 1 - RANDOM FOREST GENERATION

For our dataset I tried a couple of different configurations to the random forest generation until it improved the overall accuracy of the model. First, I run the model multiple times with different number of trees. every time adding an extra 500 trees until I was no longer able to see a substantial difference in performance.

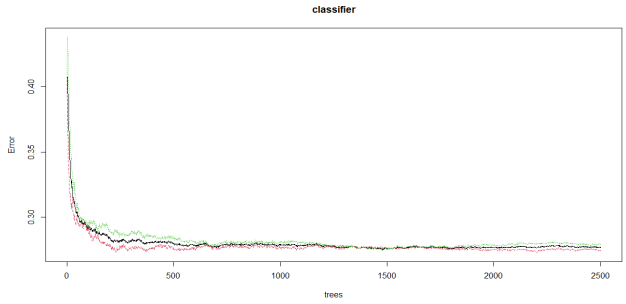



*Figure (7): Error plot for Random Forest Classification*

The image shows that as the amount of trees increased, the amount of wrong prediction decreased to a point where adding more trees would not affect the overall performance of the classifier . This finding led us to believe that the amount of trees that was optimal for the dataset was around 1500-2000 trees. To improve the performance of the model I also tweaked other parameters of the model. I set the mtry or the amount of features for each decision tree to be just 36. This way, it allows each feature to be observed at least 60 times in the whole forest. Additionally, I changed the amount of random instances passed to each tree to 7000, this way ensuring that every instance would be observed to consequently be classified by the trees a substantial amount of times in the model. All this combined improved the performance of the algorithm by 10%.

## 5. 2 DATA MINING TECHNIQUE 2 - LSTM

The LSTM model is developed in the Python program using Google Colab which is a cloud based jupyter notebook service. This proved to be very useful while developing the deep learning model as Google Colab requires no set-up or installation and also provides free access to accelerated computing resources like GPU.

### 5.2.1 DATA TRANSFORMATION FOR LSTM MODEL

Data is preprocessed and transformed before training the LSTM model. The lyrics are tokenized, and the tokenizer is applied to the training and testing lyrics so that the tokens extracted from the lyrics are converted into a sequence of integers where each word in the lyrics is given its word index. Then, padding is implemented for each sequence so that the training and testing sequences are of the same length. Zeroes are added after shorter sequences to be of the same length as the longer sequences.

The length padded training and testing sequence is 1286 which is fed into the input of the LSTM model as seen in the figure below. Note the padded sequence length equal to 1286 gives optimal 73% accuracy. But getting this padded sequence length as 1286 value is random. Because from the 15k data, the lyrics are randomly split into train and test set. So, a certain specific combination of lyrics in the train dataset gives the padded sequence length as 1286. Other times, I usually get padded sequence length as 1722 instead of 1286. And 1722 is usually fed into the input layer of the LSTM model. I get a slightly (70% -72%) lower accuracy for 1722 as input into the LSTM model. Note that the final dense_1 layer takes 1 unit because of binary classification.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1286)]            0

 embedding (Embedding)       (None, 1286, 20)          601160

 lstm (LSTM)                 (None, 1286, 15)          2160

 global_max_pooling1d (Globa  (None, 15)               0
 lMaxPooling1D)

 dense (Dense)               (None, 32)                512

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 603,865
Trainable params: 603,865
Non-trainable params: 0
_________________________________________________________________
None
```

*Figure (8): LSTM Model Architecture developed along with the input parameters*

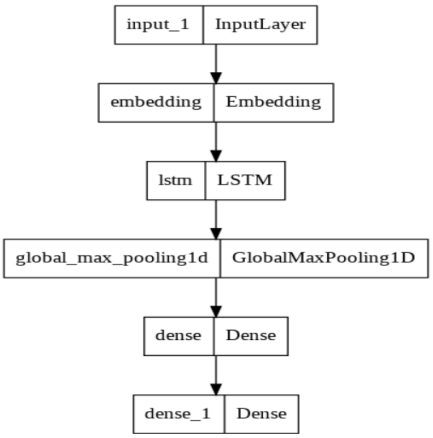



*Figure (9): Plot of LSTM Model Architecture*

### 5.2.2. ACCURACY AND LOSS GRAPHS

The LSTM model is only trained until epoch 2 because after 2 epochs the training accuracy increases at the cost of the decrease in validation accuracy and increase in validation loss. This means the model is overfitting after 2 epochs. From the two plots below it is clear that the model is not that bad because the validation loss keeps decreasing over time and the validation accuracy keeps increasing over time. Note that the validation accuracy is greater than training accuracy. This means that the model has generalized fine. It can be interpreted in two ways. Either the training data had several difficult lyrics to learn or the validation data contained easier lyrics to predict.

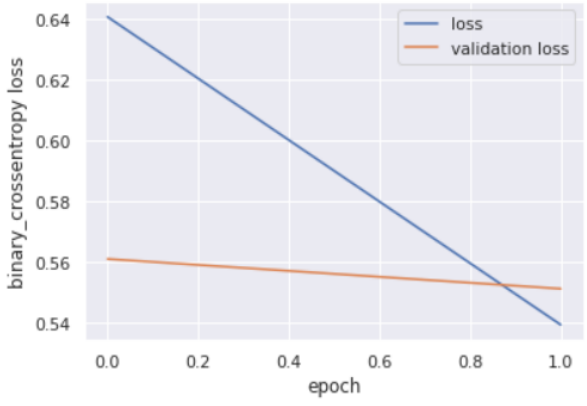



*Figure (10): Loss Plot*

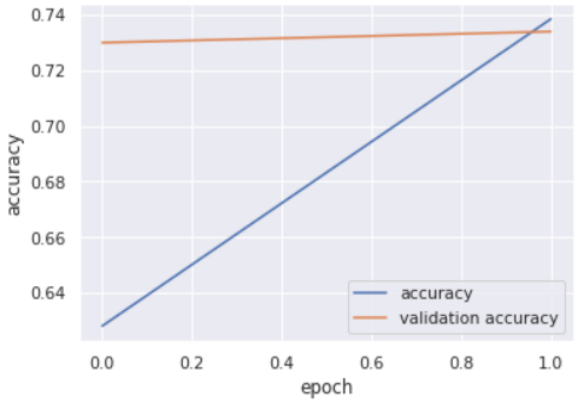



*Figure (11): Accuracy Plot*

The following tweaks to the model were tested and accuracy was recorded. Sigmoid activation is

selected instead of softmax. Using Softmax reduces the model accuracy to 49%-50%. We also selected Optimizer 'adam' instead of 'SGD'. SGD reduces the model accuracy to around 50%. Binary cross entropy loss is used to calculate loss. Besides, I chose to train the model for 2 epochs instead of 3 epochs. After 2 epochs the training accuracy increases at the cost of the decrease in validation accuracy and increase in validation loss. Also, I noted that using Bidirectional layer decreases the accuracy by 2%. The validation loss also increases along the epochs and the validation accuracy decreases. Besides, it is worth noting that if the length of the padded sequence is 1722 then the model accuracy is less by 1-2% than when the length of the padded sequence equal 1286 is obtained to be fed into the LSTM input layer. Moreover, Global max pooling operation for 1D temporal data is used in the model for dimensionality reduction. The global max pooling layer "downsampled the input representation by taking the maximum value over the time dimension" [6]. But, performance on the train set is good and continues to improve, whereas performance on the validation set improves till epoch 2 and then begins to degrade, this means the model is overfitting. This could be due to enormous features that the model needs to learn and due to complexity. To overcome this issue, Multi-task learning [1], NLP data augmentation, K-fold cross validation and feature selection [3] to feed important features into the model could be further implemented.

## 6. EVALUATION

For the purpose of evaluating the models, we decided to use the same 15k song lyrics from the dataset. Though it is important to note that some randomization was introduced during the splitting of the training and testing data set in the 80-20 splitting process. We evaluated the Random forest classifier and LSTM models developed by predicting the labels on the test set data of 3000 songs' lyrics using the confusion matrix to check the performance of the model. The confusion matrix for the LSTM model and random forest classification models can be seen in the table below. Based on the test set predictions, the final accuracy of the random forest classifier as well as LSTM model is 73%. Other Evaluation Metrics are also computed using the confusion matrix and are summarized in Table (A) below for both the models.

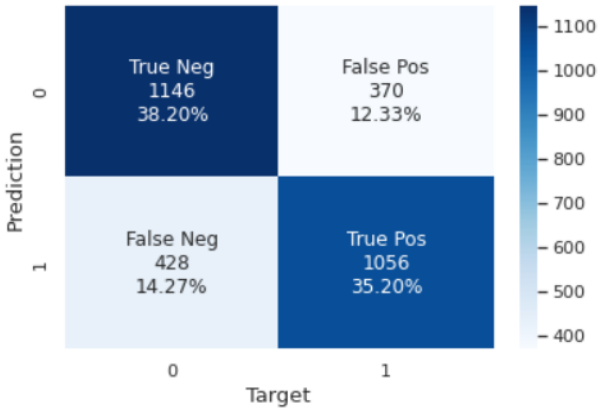



*Figure a: Confusion Matrix of LSTM*

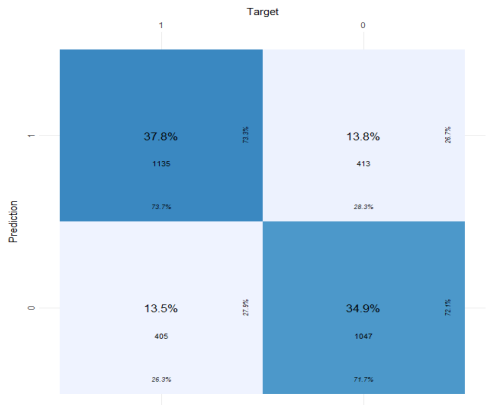



*Figure b: Confusion Matrix of RFG*

| Evaluation Metrics | LSTM Model | Random Forest |
|---|---|---|
| | Score | Score |
| Accuracy | 73% | 73% |
| Precision | 74% | 74% |
| Recall/Sensitivity | 71% | 73% |
| F-score | 73% | 72% |
| Specificity | 76% | 72% |

*Table A: Summarizes the evaluation of LSTM and Random Forest Model*

From comparing both models, it can be observed that although the accuracy and precision of both the models is the same, the F1-score and specificity of LSTM model is greater than random forest classifier by 1% and 4% respectively. Whereas, the recall/sensitivity of random forest classifier is greater than the LSTM model by 2%. With this in mind we can conclude that our model performance is good and that it does a good job at predicting most of the classes right, with a 27% error. However, there is further room for improvement.

## 7. CONCLUSION AND FUTURE WORK

In this project we have performed supervised learning for sentiment analysis of song lyrics using Random forest classification and LSTM. Both of the models performed similarly with an accuracy of about 73%. To analyze the performance of both models we used a confusion matrix,and derived the following values: recall/sensitivity, precision, F1-score and specificity. The results suggest that the performance of the algorithms are fairly decent, predicting most of the songs overall net sentiment correct with a 27% error. To improve the accuracy of the models in some future work, we would use modern NLP state-of-the-art techniques based on context like transformer models from the HuggingFace [1] Library, using lyrics text and audio concurrently [5] or using audio spectrogram [3] and other features other than words to fit the model. For example, extracting the other attribute labels like acousticness, danceability, energy, instrumentalness, liveness, loudness, mode, speechiness and tempo from spotify API and take them into consideration for training the model [4,5]. For each of these factors we could add a weight that would determine which feature will impact the overall net sentiment of a song the most, allowing the model to not only take into account the words but also the audio features of the songs. Additionally, we could use n-grams to better understand the overall meaning of words when they are combined together. Also, working with more data could definitely help increase the overall performance of the models.

## REFERENCES


[1.] Agrawal Y, Shanker RGR, Alluri V. Transformer-based approach towards music emotion recognition from lyrics [Internet]. SpringerLink. Springer International Publishing; 2021 [cited 2022May1]. Available from: https://link.springer.com/chapter/10.1007/978-3-030-72240-1_12

[2.] Seo Y-S, Huh J-H. Automatic emotion-based music classification for supporting intelligent IOT Applications [Internet]. MDPI. Multidisciplinary Digital Publishing Institute; 2019 [cited 2022Apr20]. Available from: https://doi.org/10.3390/electronics8020164

[3.]Chen C, Li Q. A multimodal music emotion classification method based on multifeature combined network classifier [Internet]. Mathematical Problems in Engineering. Hindawi; 2020 [cited 2022Apr20]. Available from: https://doi.org/10.1155/2020/4606027

[4.] Web API reference: Spotify for developers [Internet]. Spotify for Developers. WEB API Reference; 2019 [cited 2022May7]. Available from: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features

[5.] Pyrovolakis K, Tzouveli P, Stamou G. Multi-modal song Mood detection with deep learning [Internet]. MDPI. Multidisciplinary Digital Publishing Institute; 2022 [cited 2022Apr22]. Available from: https://www.mdpi.com/1424-8220/22/3/106

[6.]Team K. Keras Documentation: Keras Layers API [Internet]. Keras. 2015 [cited 2022Apr22]. Available from: https://keras.io/api/layers/

## CODE

We have partitioned the code into sections which are: plots, data preprocessing, random forest classification and LSTM. The code can be found here: Sentiment Analysis in R and python. The project uses both python and R. Python was used for LSTM, and R was used for preprocessing the data and to create the random forest model. To run the R code, make sure you first open the file call sentiment_analysis_in_R.RPROJ. After that each folder will have a main script file where we have purposely separated each step of this project to have a cleaner and loosely couple program to easily navigate through the whole project.

### DATA PREPROCESSING

In this part of the code, we will focus mainly on how we manage to preprocess the dataset. The following code can be found in the preprocess_data folder, in the script call main.R.

```
needed_packages <- c("dplyr", "stringr", "tidytext",
"tidyr","textdata", "tm", "SnowballC", "caTools", "rlang", "gt",
"stopwords","sentimentr", "tidytext", "magrittr", "textstem","qdap",
"doParallel", "devtools", "cld2", "superml")

#install packages in case they are not install yet (uncomment the
next lines)
#install.packages(needed_packages)
#devtools::install_github("saraswatmks/superml")
#install.packages("superml", dependencies=TRUE)

#load packages
lapply(needed_packages, require, character.only = TRUE)

source("./preprocessing_data/remove_redundancies_and_bad_words.R")
source("./preprocessing_data/labeling_data.R")
source("./preprocessing_data/tf_idf_model.R")
source("./preprocessing_data/bag_of_words.R")

#load dataset
dataset <- read.csv("./data/labeled_lyrics_cleaned.csv")

#additional swear words found by preliminary analysis
```

```
swears <- read.csv("./data/swear_words.csv")

#additional stopwords found by preliminary analysis
additional_stopwords <- read.csv("./data/additional_stop_words.csv")
#change the names of the columns and set the labels >= 0.67 to be 1
#and labels <= 0.33 to be 0
dataset  <- dataset  %>%
  transmute(artist = artist
            ,lyric = data$seq
            ,title = song
            ,rating = ifelse(label >= 0.67 ,
                             1 ,
                             ifelse(label <= 0.33, 0,  label)))

#remove any songs that do not have a label rating of 0 or 1
dataset  <- dataset %>%
  filter(rating == 1 | rating == 0)
```

The following line uses a function to clean the lyric column

```
#cleans the lyric column by using this function
dataset  <- remove_redundancies_and_bad_Words(dataset)
```

The function can be found in  a different r script which is call remove_redundancies_and_bad_words.R. The definition for the function is the following:

```
remove_redundancies_and_bad_Words <- function(dataset) {

  #detect non-english songs to remove it
  non_english_songs_remover <- ifelse(detect_language(dataset$lyric)
!= "en",TRUE,FALSE)

  dataset = dataset[!non_english_songs_remover,]

  # function to expand contractions in an English-language source
  fix.contractions <- function(doc) {
    # "won't" is a special case as it does not expand to "wo not"
```

```
    doc <- gsub("won't", "will not", doc)
    doc <- gsub("can't", "can not", doc)
    doc <- gsub("n't", " not", doc)
    doc <- gsub("i'll", "i will", doc)
    doc <- gsub("i've", "i have", doc)
    doc <- gsub("i'm", "i am", doc)
    doc <- gsub("i'd", "i would", doc)
    doc <- gsub("in'", "ing", doc)
    doc <- gsub("woulda", "would have", doc)

    # 's could be 'is' or could be possessive: it has no expansion
    doc <- gsub("'s", "", doc)
    doc <- gsub("'cause", " because", doc)
    return(doc)
  }

  # fix (expand) contractions
  dataset$lyric <- sapply(dataset$lyric, fix.contractions)

  #remove songs with few lyrics
  few_lyrics_remover = ifelse(nchar(dataset$lyric) < 350, TRUE,
FALSE)
  dataset = dataset[!few_lyrics_remover,]

  #converts words to lowercase,lemmatize words
  #replaces extra contractions, removes numbers and punctuations
  dataset <- dataset %>%
    mutate(lyric = sapply(lyric, tolower)) %>%
    mutate(lyric = replace_contraction(lyric)) %>%
    mutate(lyric = removePunctuation(lyric)) %>%
    mutate(lyric = removeNumbers(lyric)) %>%
    mutate(lyric = lemmatize_strings(lyric)) %>%
    mutate(lyric = sapply(lyric, tolower))


  #traverses the data and extracts every profanity word in each song
  dataset <- dataset %>%
    mutate(lyric = extract_profanity_terms(get_sentences(lyric),
                                           profanity_list =
```

```
swears$swear_words))

  #saving the swear words into this vector
  swear_words_list <- unlist(dataset$lyric$profanity,
                             recursive = TRUE,
                             use.names = TRUE)

  #removes duplicates of same word
  swear_words_list <- paste(unique(swear_words_list))

  #passes all the stop words to this vector so that we can later
remove it from
  #our data
  stopwords_list = paste(stopwords('en'), collapse = '\\b|\\b')
  stopwords_list = paste0('\\b', stopwords_list, '\\b')

  #removes stop words and additional white spaces
  dataset <- dataset %>%
    mutate(lyric = str_remove_all(lyric, stopwords_list)) %>%
    mutate(lyric = stripWhitespace(lyric))

  #removes the swear words that have in common with the list
  dataset <- dataset %>%
    mutate(lyric = removeWords(lyric$sentence, swear_words_list))

  #remove additional stop words
  dataset <- dataset %>%
    mutate(lyric = removeWords(lyric,
additional_stopwords$stop_words))

  return(dataset)
}
```

Going back to main.r script of the preprocess_data folder.

```
#The following steps are to get a even amount of positive and negative
songs
```

```r
#get only overall positive songs
positive_songs <- dataset  %>%
  filter(rating == 1)

#get only overall negative songs
negative_songs <- dataset %>%
  filter(rating == 0)

#extract 7500 songs from each postive and negative data frame
positive_songs <- sample_n(positive_songs , 7500)
negative_songs <- sample_n(negative_songs, 7500)

#to get a perfectly balanced amount of positive and negative songs
dataset <- rbind(positive_songs, negative_songs)

#shuffle the data to randomize the positions of postive and negative
songs
dataset = dataset[sample(1:nrow(dataset)), ]

#save the cleaned dataset to use it later
write.csv(dataset,
          "./data/cleaned_15k_dataset.csv",
          row.names = FALSE)

#general information
dim(dataset) # (15000, 4)  --> 15000 rows, 4 columns
str(dataset)
summary(dataset)

# look at some example texts randomly from the dataset
dataset %>% select(lyric) %>% sample_n(4) %>% pull()

unique(dataset$artist) # 1000+ unique artists in the dataset
dataset %>% View()
```

## DATA TRANSFORMATION FOR RANDOM FOREST

The following line uses a defined unction to vectorize the strings.

```
#creating tf_idf vector to work with the classifier
tf_idf_dataset <- tf_idf_vectorizer(dataset)
```

The function can be found in  a different r script which is call tf_idf_model.R. The definition for the function is the following:

```
#creating the tf_idf_vector model
tf_idf_vectorizer <- function(dataset_original) {


  #vcorpus is a data structure that will help cleaning
  #the text so we can work with our data ussing this vector technique
  corpus_dataset = VCorpus(VectorSource(dataset_original$lyric))
  #puts all the words into lowercases
  corpus_dataset = tm_map(corpus_dataset,
content_transformer(tolower))
  #removes white spaces and extra spaces
  corpus_dataset = tm_map(corpus_dataset, stripWhitespace)

  #converts corpus data structure to a vector
  vector = data.frame(lyric = sapply(corpus_dataset, as.character),
                      stringsAsFactors = FALSE)

  #creates tf-idf vectorizer.It only takes into account features that
appear at
  #least .0050 in entire data to reduce dimensions
  tfv <- TfIdfVectorizer$new(min_df = 0.0050, remove_stopwords =
FALSE)

  #parallel processing to speed up the process
  cl <- makePSOCKcluster(detectCores() - 1)
  registerDoParallel(cl)

  #fits the data with the parameters chosen using the formula for
```

```
tf-idf
  tfv$fit(vector$lyric)

  #tranforms the values into tf-idf values
  tf_matrix <- tfv$transform(vector$lyric)

  stopCluster(cl)

  #we use as.data.frame to transform the matrix to a data frame
  dataset = as.data.frame(as.matrix(tf_matrix))

  return(dataset)
}
```

Next in the main.R script of the preprocess_data folder:

```
#copy the dependent feature to the vectorizer data
tf_idf_dataset$rating = dataset$rating

#Encoding the dependent feature as factor
tf_idf_dataset$rating = factor(tf_idf_dataset$rating,
                                    levels = c(0, 1))

#saving the tf_idf data to user it later
write.csv(tf_idf_dataset,
          "./data/tf_idf_dataset.csv",
          row.names = FALSE)
```

The following line uses another defined function call bag_of_words:

```
#creating a bag of words vector to work with the classifier
bag_of_word_dataset <- bag_of_words(dataset)
```

The definition for that function can be found in the script call bag_of_words.R in the preprocess_data folder. The function looks like the following:

```
#creating bag of words
bag_of_words <- function(dataset_original) {

  #cleaning up the data
  #vcorpus is a data structure that will help cleaning
  #the text so we can work with the data using the bag of words model
  corpus_dataset = VCorpus(VectorSource(dataset$lyric))

  #puts all the words in lowercases
  corpus_dataset = tm_map(corpus_dataset,
content_transformer(tolower))
  #removes white spaces and extra spaces
  corpus_dataset = tm_map(corpus_dataset, stripWhitespace)


  #Creating the Bag of Words model.
  dtm = DocumentTermMatrix(corpus_dataset)

  #removes the words that do not appear frequently,
  #filter non-frequent words that
  #don't add any meaning to our data
  dtm = removeSparseTerms(dtm, 0.9951)

  #creating a vector to work with the algorithm
  #we use as.data.frame to transform in this case a matrix to a data
frame
  bag_of_word_dataset = as.data.frame(as.matrix(dtm))

  return(dataset)
}
```

Finally, in the main.R script of the preprocess_data folder:

```
#copy the dependent feature to the vectorizer data
bag_of_word_dataset$rating = dataset$rating
```

```
#Encoding the dependent feature as fsactor
bag_of_word_dataset$rating = factor(bag_of_word_dataset$rating,
                                    levels = c(0, 1))

#saving the bag of words data to use it later
write.csv(bag_of_word_dataset,
          "./data/bag_of_words_dataset.csv",
          row.names = FALSE)
```

That would conclude the processing data part of the project. Now to better see and understand the data that was cleaned, there is a section in the repository call plots, where we have the scripts to create the tables and graphs to display the data. The followin code is from the main.R script which is located on the folder call plots:

```
#loading and installing packages
needed_packages <- c("dplyr", "stringr", "tidytext",
"tidyr","textdata","tm", "SnowballC", "caTools","rlang", "gt",
"stopwords","sentimentr", "htmlwidgets", "webshot", "kableExtra",
"wordcloud", "wordcloud2", "magrittr", "ggplot2", "textstem")

#uncomment in case these packages are not install
#install.packages(needed_packages)

lapply(needed_packages, require, character.only = TRUE)

source("./plots/display_tokenize_data.R")
source("./plots/display_vector_technique.R")
source("./plots/display_cleaned_data.R")

#loading the data
dataset <- read.csv("./data/cleaned_15k_dataset.csv")

#copying the data to a new table
dataset1 <- dataset
```

The next line uses a function which is placed in another script call display_cleaned_data.R.

```
#display clean dataset in a table
create_table_to_display_clean_dataset(dataset)
```

The definition for this function is the following:

```
create_table_to_display_clean_dataset <- function(dataset) {

  data_temp =  dataset %>%
    sample_n(4)

  data_temp %>%
    gt() %>%
    tab_header(
      title = md("**Clean dataset**"),
      subtitle = md("table shows a few lyrics after having been
cleaned")
    )
}
```

Continuing in the main file script for the folder plots:

```
#select artists with most songs to display the frequency of
sentiments
selected_artist <- dataset %>%
  group_by(artist) %>%
  count(artist) %>%
  ungroup() %>%
  filter(n >= 34)

#filter songs by artists
selected_artists_song <- dataset %>%
  filter( artist %in% selected_artist$artist)

#display tokenize data
create_graph_to_display_frequency_of_sentiments(selected_artists_song
)
```

```
#loading tf_idf vectorization method
tf_idf_model <- read.csv("./data/tf_idf_dataset.csv")

#display tf_idf
create_table_for_vector_technique(tf_idf_model)

#loading bag_of_word vectorization method
bag_of_words_model <- read.csv("./data/bag_of_words_dataset.csv")

#display bag_of_words
create_table_for_vector_technique(bag_of_words_model)

#vector to corpus data structure to clean data in an efficient manner
Corpus_data <- Corpus(VectorSource(dataset1$lyric))

Song_Lyrics_dtm <- TermDocumentMatrix(Corpus_data)
dtm_m <- as.matrix(Song_Lyrics_dtm)

#display common words in a table when data is cleaned
dtm_v <- sort(rowSums(dtm_m), decreasing = TRUE)
dtm_d <- data.frame(word = names(dtm_v), freq = dtm_v)
kable(head(dtm_d, 10), col.names = c("Word", "Frequency"), row.names
= FALSE,
      caption = "Table 1: Most Common Terms (Cleaned data)", align =
"c") %>%
  kable_styling(full_width = F)

#-----------------------------------------------------------------------
# Wordcloud
#-----------------------------------------------------------------------

#displays a wordcloud of the cleaned data
webshot::install_phantomjs(force = TRUE)

wordcloud2(dtm_d, fontFamily = "Comic Sans", size = 1.2)
```

Some other functions used to display graphs are:

```
create_table_for_vector_technique <- function(dataset) {
```

```
  data_temp =  dataset[1:20,1:11]

  data_temp$rating = dataset[1:20, ncol(dataset)]

  data_temp %>%
    gt() %>%
    tab_header(
      title = md("**Vectorization technique**"),
      subtitle = md("table shows the first 20 songs with a few
selected words")
    )
}
```

```
create_graph_to_display_frequency_of_sentiments <- function(dataset)
{

  #separating each word from a single row/lyric while
  #keeping track of the lyric
  tokenized_dataset <- dataset %>%
    group_by(artist) %>%
    mutate(linenumber = row_number()) %>%
    ungroup() %>%
    unnest_tokens(word, lyric)

  #we get the words that have in common with bing and use the line
number to
  #to keep track of what song that word belongs to.After that,
  #we separate the sentiments positive and negative by
  #using pivot wider and we count the frequency of positive and
negative sentiments
  prepare_tokenized_dataset <- tokenized_dataset %>%
    inner_join(get_sentiments("bing")) %>%
    count(artist, song_number = linenumber, sentiment) %>%
    pivot_wider(names_from = sentiment, values_from = n, values_fill
= 0) %>%
```

```
    mutate(sentiment = positive - negative)

  ggplot(prepare_tokenized_dataset, aes(song_number, sentiment, fill=
artist)) +
    geom_col(show.legend = FALSE) +
    facet_wrap(~artist, ncol = 4, scales= "free_x")

}
```

## DATA TECHNIQUE 1- RANDOM FOREST

This part of the code can be found in the folder call random_forest_classifier, in the only script that is available in there call main.R. In this part, we create the random forest model and evaluate the technique to check the performance of it.

```
needed_packages <- c("dplyr", "stringr", "tidytext", "tidyr", "textdata",
"tm","caTools","gt","tidytext","magrittr","randomForest","qdap","doParalle"
, "superml", "devtools", "ROCR", "cvms","caret", "ggimage", "rsvg")

#install packages in case they are not install yet
#install.packages(needed_packages)

#load packages
lapply(needed_packages, require, character.only = TRUE)

#loading the preprocessed data
tf_idf_dataset <- read.csv("./data/tf_idf_dataset.csv")

#Encoding the target feature as factor
tf_idf_dataset$rating = factor(tf_idf_dataset$rating,
                                    levels = c(0, 1))

#splitting the data into training set and test set.
#split the data in 80% for training set and 20% for the test set
split = sort(sample(nrow(tf_idf_dataset), nrow(tf_idf_dataset)*.8))
training_set = tf_idf_dataset[split,]
test_set = tf_idf_dataset[-split,]
```

```
#run model with parallel processing using doparallel library to speed up
the
#Processss
cl <- makePSOCKcluster(detectCores() - 1)
registerDoParallel(cl)

start.time <- proc.time()

#training model using random forest classifier
classifier = randomForest(x = training_set[-ncol(training_set)],
                          y = training_set$rating,
                          ntree = 2501,
                          sampsize = 7000,
                          mtry = 37,
                          type = "classification",
                          do.trace = 25)

stop.time <- proc.time()

run.time <- stop.time - start.time
print(run.time)

stopCluster(cl)

cl <- makePSOCKcluster(detectCores() - 1)
registerDoParallel(cl)

#get better settings for random forest classifier, takes a lot  of time to
compute values
classifier_best_setting = train(y = training_set$rating,
                                x = training_set[-ncol(training_set)],
                  data = training_set[-ncol(training_set)],
                  method = "rf")

stopCluster(cl)

#predicting test results
y_pred_test_set = predict(classifier, newdata = test_set[-ncol(test_set)])

#Making the Confusion Matrix to compare results
confusion_matrix = table("target" = test_set$rating, "predicted" =
y_pred_test_set)
```

```
#Accuracy shows the amount of correctly predictions
accuracy_val = multiply_by(divide_by(confusion_matrix[1] +
confusion_matrix[4],
                   nrow(test_set))
                  , 100)
cat(accuracy_val, "% lyrics label were predicted correctly", sep = '')

#Precision shows the amount of predicted positive songs that were correctly
precision_val = multiply_by(divide_by(confusion_matrix[1],
                                      confusion_matrix[2] +
confusion_matrix[1])
                            , 100)

cat(precision_val, "% of lyrics that aroused overall positive feelings from
the
predicted positives, were predicted correctly", sep = '')

#sensitivity shows the amount of positive overall examples that were
predicted
#accurately
sensitivity = multiply_by(divide_by(confusion_matrix[1],
                                    confusion_matrix[3] +
confusion_matrix[1])
                          , 100)

cat(sensitivity, "% of lyrics that aroused positive feelings only,
    were predicted accurrately", sep = '')

#fp rate shows the amount of negative values predictive incorrectly
fp_rate = multiply_by(divide_by(confusion_matrix[2],
                                confusion_matrix[2] + confusion_matrix[4])
                      , 100)

cat(fp_rate, "% of lyrics that were predicted as arousing positive
feelings,
    were predicted wrongly", sep = '')

#specificity show the amount of negative feeling songs that were predicted
correctly
specificity = multiply_by(divide_by(confusion_matrix[4],
                         (confusion_matrix[2] + confusion_matrix[4])),
                         100)
cat(specificity, "% of lyrics that aroused overall positive feelings from
```

```
the
predicted positives, were predicted correctly", sep = '')

#mean of precision and recall
f_score = divide_by(multiply_by(2, multiply_by(precision_val,
sensitivity)),
                    precision_val + sensitivity)

cat(f_score, "% harmonic mean of precision and recall", sep = '')

#plot confusion matrix for test set
cfm <- as_tibble(confusion_matrix)


plot_confusion_matrix(cfm,
                      target_col = "target",
                      prediction_col = "predicted",
                      counts_col = "n")

#check how the amount of trees affect the accuracy of the predictions
plot(classifier)
```

## DATA TECHNIQUE 2 - LSTM

```
# mounting the drive
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
[ ]
# Importing libraries and modules
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ]
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data/cleaned_15k_dataset.csv', encoding='latin-1')
```

```
[ ]
# printing the dataframe
df.head(10)
```

| | artist | lyric | title | rating |
|---|---|---|---|---|
| 0 | Hillsong | i will love you lord my strength i belong to y... | I Will Love | 0 |
| 1 | Vera Lynn | when the light go on again all over the world ... | When the Lights Go on Again | 0 |
| 2 | Georgia Mass Choir | when you be down and in despair do not be unea... | Hold On, Help Is on the Way | 0 |
| 3 | Richard Wyands | let build a stairway to the star and climb tha... | Stairway to the Stars | 0 |
| 4 | Echo & the Bunnymen | show me something that i have not see before s... | Proxy | 1 |
| 5 | Tripping Daisy | she the melt pot of summer she a balloon with ... | Stella Is a Planet | 0 |
| 6 | Brian McKnight | nelly look just ai not the same when you slid... | All Night Long | 1 |
| 7 | Lenny Kravitz | i be you and you be me why that such a mystery... | Believe | 0 |
| 8 | Billy Bragg | on monday i wish it be tuesday night so i can ... | Wishing the Days Away [Ballad Version] | 0 |
| 9 | Betty Elders | i have see that kind of hurt before you keep i... | Crayons | 0 |

```
[ ]
# drop the unnecessary columns
df.drop('artist', axis=1, inplace=True)
df.head(10)
```

| | lyric | title | rating |
|---|---|---|---|
| 0 | i will love you lord my strength i belong to y... | I Will Love | 0 |
| 1 | when the light go on again all over the world ... | When the Lights Go on Again | 0 |
| 2 | when you be down and in despair do not be unea... | Hold On, Help Is on the Way | 0 |
| 3 | let build a stairway to the star and climb tha... | Stairway to the Stars | 0 |
| 4 | show me something that i have not see before s... | Proxy | 1 |
| 5 | she the melt pot of summer she a balloon with ... | Stella Is a Planet | 0 |
| 6 | nelly look just ai not the same when you slid... | All Night Long | 1 |
| 7 | i be you and you be me why that such a mystery... | Believe | 0 |
| 8 | on monday i wish it be tuesday night so i can ... | Wishing the Days Away [Ballad Version] | 0 |
| 9 | i have see that kind of hurt before you keep i... | Crayons | 0 |

```
import string, re
import itertools
import nltk
import plotly.offline as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from wordcloud import WordCloud,STOPWORDS
```

```
[ ]  1 # drop the unnecessary columns
     2 df.drop('title', axis=1, inplace=True)
     3 df.head(10)
```

| | lyric | rating |
|---|---|---|
| **0** | i will love you lord my strength i belong to y... | 0 |
| **1** | when the light go on again all over the world ... | 0 |
| **2** | when you be down and in despair do not be unea... | 0 |
| **3** | let build a stairway to the star and climb tha... | 0 |
| **4** | show me something that i have not see before s... | 1 |
| **5** | she the melt pot of summer she a balloon with ... | 0 |
| **6** | nelly look just ai not the same when you slid... | 1 |
| **7** | i be you and you be me why that such a mystery... | 0 |
| **8** | on monday i wish it be tuesday night so i can ... | 0 |
| **9** | i have see that kind of hurt before you keep i... | 0 |

```
[ ]  1 # veryfying the sentiment values
     2 # 1 is positive sentiment and 0 is negative sentiment
     3 df['rating'].value_counts()
     4
     5 #0    7500
     6 #1    7500
```

```
0    7500
1    7500
Name: rating, dtype: int64
```

```
[ ]  1 lens = df['lyric'].str.len()
     2 print(lens.mean())
     3 print(lens.std())
     4 print(lens.min())
     5 print(lens.max())
```

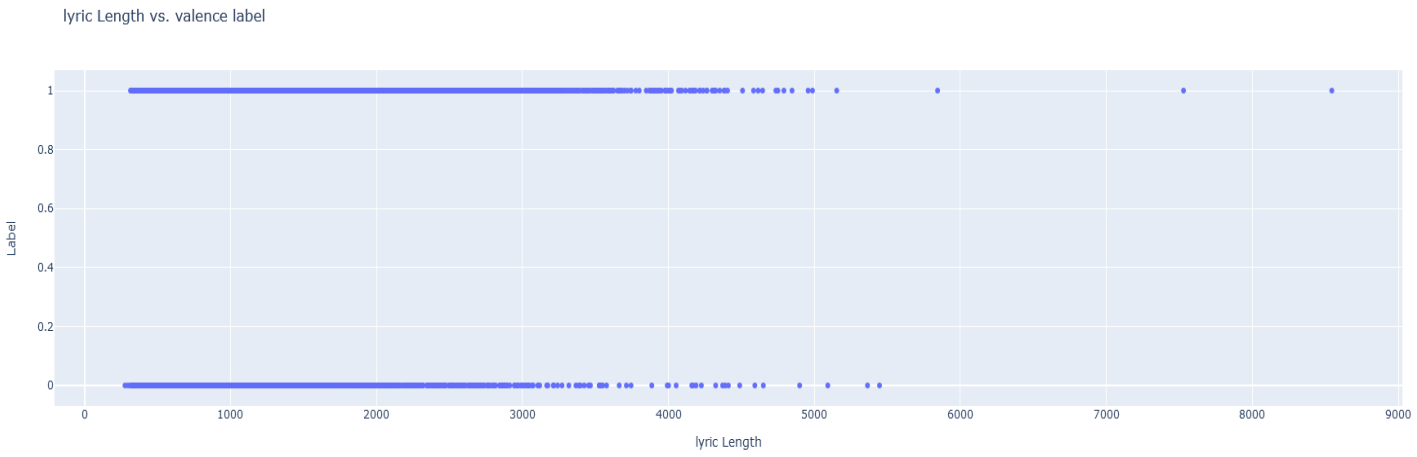
```
1048.197
596.8841662932017
277
8545
```

.

```
1 print(df.dropna().shape)
```

```
(15000, 2)
```

```
1 df['senLen'] = df['lyric'].apply(lambda x: len(x))
2 data = df.sort_values(by='senLen')
3 plot = go.Scatter(x = data['senLen'], y = data['rating'], mode='markers')
4 lyt = go.Layout(title="lyric Length vs. valence label", xaxis=dict(title='lyric Length'),yaxis=dict(title='Label'))
5 fig = go.Figure(data=[plot], layout=lyt)
6 py.iplot(fig)
```

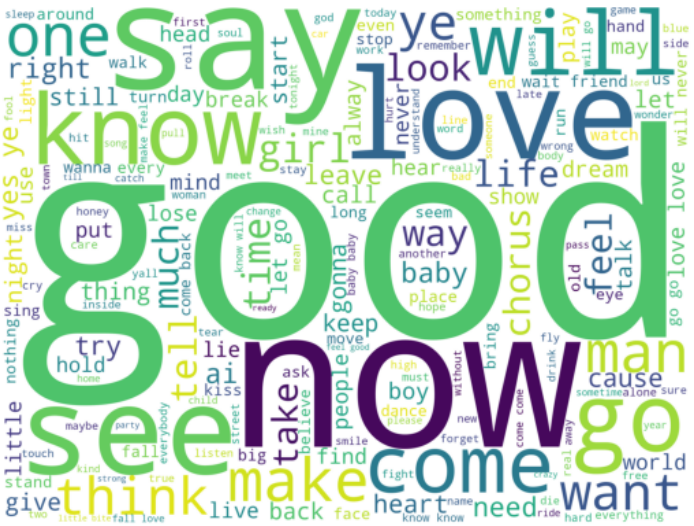


```
1 from wordcloud import WordCloud,STOPWORDS
```

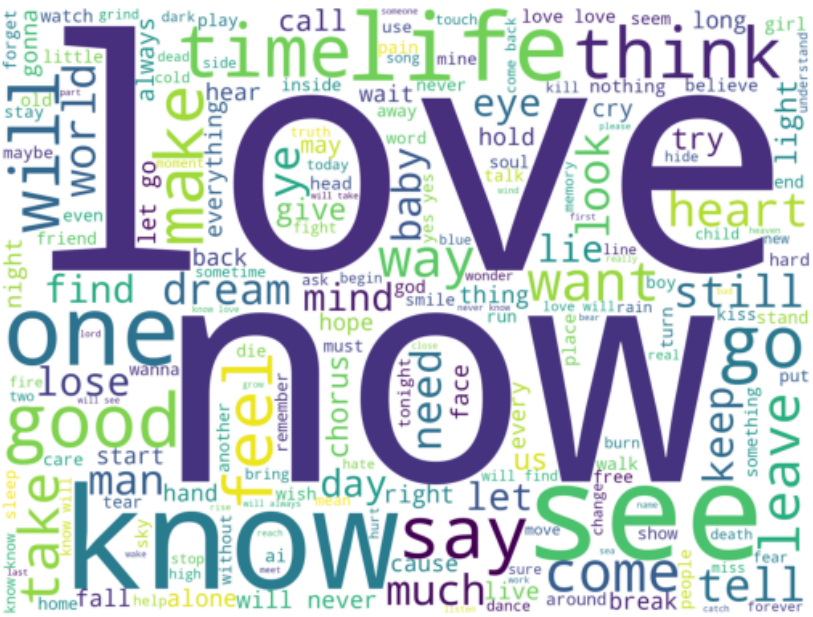
```
1 df.head(5)
```

| | lyric | rating |
|---|---|---|
| **0** | i will love you lord my strength i belong to y... | 0 |
| **1** | when the light go on again all over the world ... | 0 |
| **2** | when you be down and in despair do not be unea... | 0 |
| **3** | let build a stairway to the star and climb tha... | 0 |
| **4** | show me something that i have not see before s... | 1 |

```
1 df_pos = df[ df['rating'] == 1]
2 df_pos = df_pos['lyric']
3
4 wordcloud1 = WordCloud(stopwords=STOPWORDS,
5                        background_color='white',
6                        width=2000,
7                        height=1500
8                        ).generate(" ".join(df_pos))
9 plt.figure(1,figsize=(9, 9))
10 plt.imshow(wordcloud1)
11 plt.axis('off')
12 plt.show()
```



```python
df_neg = df[ df['rating'] == 0]
df_neg = df_neg['lyric']

wordcloud1 = WordCloud(stopwords=STOPWORDS,
                       background_color='white',
                       width=2000,
                       height=1500
                       ).generate(" ".join(df_neg))
plt.figure(1,figsize=(9, 9))
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()
```



```
sns.set(style="darkgrid")
b = sns.countplot(x='rating',
                  data = df.drop(['lyric'], axis=1))
b.axes.set_title('Sentiment (target variable) distribution')
b.set_xlabel("Label")
b.set_ylabel("Count")
plt.show()
```


The 80-20 split into training and testing dataset is random during each run.

```
[ ]  df.lyric = df.lyric.astype(str)
```

```
[ ]  # splitting the data into training and testing data
     from sklearn.model_selection import train_test_split
```

```
[ ]  x_train, x_test, y_train, y_test = train_test_split(df['lyric'].values, df['rating'].values, test_size=0.20)  # 80-20 training andtest split
     # split the lyric text into x_train and x_test and rating (valence) into y_train and y_test.
```

```
[ ]  # Check the x_train data after split
     print('Lyrics: ', x_train[0])
```

```
sentiment Text:  people be look but they do not know what to do it the time of the season for the people like you come back tomorrow show the sc
```

```
[ ]  # Check the y_train data after split
     print('Valence: ', y_train[0])
```

```
Valence:  0
```

```
[ ]  # Check the x_test data  after split
     print('Lyrics: ', x_test[0])
```

```
Lyrics:  some people can get a thrill knit sweater and set still that okay for some people who do not know they be alive some people can thrive
```

```
     # Check the y_test data  after split
     print('Valence: ', y_test[0])
```

```
Valence:  0
```

```
[ ]  1 # converting the strings into integers using Tokenizer
     2 from tensorflow.keras.preprocessing.text import Tokenizer
     3 from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
[ ]  1 # instantiating the tokenizer
     2 MAX_VOCAB = 90000000 # is basically max features. How many unique words to use i.e., num or rows in embedding vector)
     3 tokenizer = Tokenizer(num_words= MAX_VOCAB)
     4 tokenizer.fit_on_texts(x_train) # fit on input x_train data
```

```
[ ]  1 # Checking the word index for the length of vocab we have
     2 WordIndex = tokenizer.word_index
     3
     4 # Calculate the vocabulary size of the dataset
     5 Vocab_size = len(WordIndex)
     6 print('The size of datatset vocab is: ', Vocab_size)
```

```
The size of datatset vocab is:  30057
```

**Tokenization:**

I love a cat = ["I", "love", "a", "cat"] = [1, 2, 3, 4]

I have a cat = ["I", have", "a", "cat"] = [1, 5, 3, 4]

I have a cat and dog = ["I", have", "a", "cat", "and", "dog"] = [1, 5, 3, 4, "6","7"]

```
[ ]  1 # Convert Train data and Test data text into sequences
     2 train_seq = tokenizer.texts_to_sequences(x_train)
     3 test_seq = tokenizer.texts_to_sequences(x_test)
     4 print('Training sequence: ', train_seq[0])
     5 print('Testing sequence: ', test_seq[0])
```

```
Training sequence:  [163, 4, 75, 29, 49, 13, 9, 26, 33, 5, 13, 8, 3, 46, 15, 3, 853, 25, 3, 1
Testing sequence:  [108, 163, 19, 20, 7, 595, 10496, 3372, 6, 333, 111, 14, 546, 25, 108, 163
```

**Padding:**

I love a cat = ["I", "love", "a", "cat"] = [1, 2, 3, 4] = [1, 2, 3, 4, 0, 0]

I have a cat = ["I", have", "a", "cat"] = [1, 5, 3, 4] = [1, 5, 3, 4, 0, 0]

I have a cat and dog = ["I", have", "a", "cat", "and", "dog"] = [1, 5, 3, 4, "6","7"]

Notice that the train and test sequences are of unequal length so, now apply padding so that the train and test sequences are of equal length.

```
[ ]  1 # Padding the training sequences to get equal length sequence to use sequences of equal length
     2 pad_train = pad_sequences(train_seq)
     3 padded_seq_len = pad_train.shape[1]
     4 print('The length of training sequence is: ', padded_seq_len)  # Note the 1286 gives optimal 73% accuracy. Because getting this 1286 is random. Beccause of the 15k data, the lyrics are randomly split into train and test.
     5 # so the a certain specific combination of lyrics in train dataset gives the padded_sequence_length as 1286.
     6 # Other times, I usually get padded_sequence_length as 1722 in stead of 1286. And 1722 is usually fed into input layer of the LSTM model.
```

```
The length of training sequence is:  1286
```

```
[ ]  1 # Padding the test sequences to get equal length sequence to use sequences of equal length
     2 pad_test = pad_sequences(test_seq, maxlen=padded_seq_len)
     3 print('The length of testing sequence is: ', pad_test.shape[1])
```

```
The length of testing sequence is:  1286
```

Run till above code cell and then execute the load model to predict the unknown lyrics, so that you can run the saved model from google drive.

```
# Model using Keras API

from tensorflow.keras.layers import Input, Dense, Embedding, LSTM, GlobalMaxPooling1D
from tensorflow.keras.models import Model # intiate the model

# Layers used in this model are : Input, Dense, Embedding, LSTM, GlobalMaxPooling1D
# https://keras.io/api/layers/  --> Kera layers documentation
```

```
[ ]
Embedding_layer_dimension = 20
LSTM_layer_dimension = 15

input_layer = Input (shape=(padded_seq_len, ))
output = Embedding(Vocab_size + 1, Embedding_layer_dimension)(input_layer)    #  Vocab_size + 1 because the indexing of the words in vocabulary (Vocab_size) start from 1 not 0
output = LSTM(LSTM_layer_dimension, return_sequences=True)(output)
output = GlobalMaxPooling1D()(output) # Global max pooling operation for 1D temporal data. Downsamples the input representation by taking the maximum value over the time dimension.
output = Dense(32, activation='relu')(output)
output = Dense(1, activation='sigmoid')(output) # Dense layer with class 1 because the sentiment valence is either positive or negative
# And because of binary classification 'sigmoid' activation is more suitable.

```

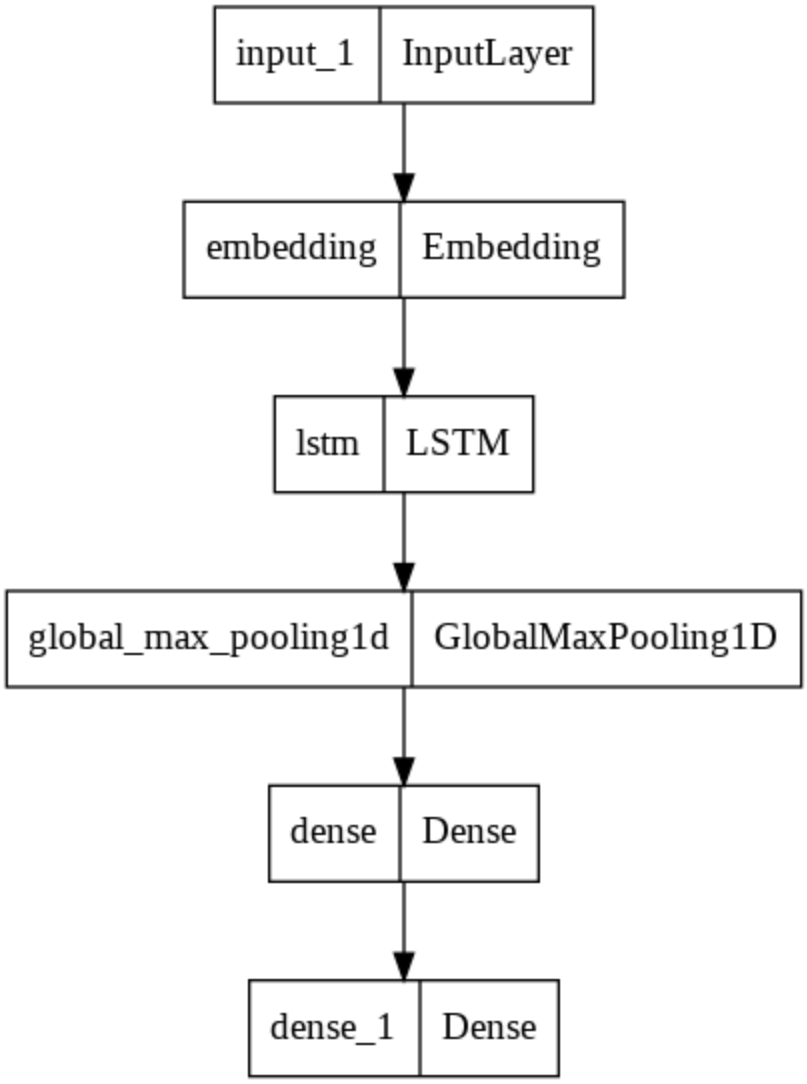
```
[ ]
# feed input and output to the model for model intiation.
model = Model(input_layer,output)
```

```
[ ]
# compiling the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# 'binary_crossentropy' loss because of binary classification (pos and neg)
```

```
[ ]   1 print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1286)]            0

 embedding (Embedding)       (None, 1286, 20)          601160

 lstm (LSTM)                 (None, 1286, 15)          2160

 global_max_pooling1d (Globa  (None, 15)               0
 lMaxPooling1D)

 dense (Dense)               (None, 32)                512

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 603,865
Trainable params: 603,865
Non-trainable params: 0
_________________________________________________________________
None
```

```
[ ]  1 from tensorflow.keras.utils import plot_model
     2 plot_model(model)
```




```
[ ]  1 # training the model
     2 r = model.fit(pad_train, y_train,
     3                validation_data=(pad_test, y_test),
     4                epochs=2,
     5                shuffle=True,
     6                verbose=1)
```

```
Epoch 1/2
375/375 [==============================] - 200s 527ms/step - loss: 0.6407 - accuracy: 0.6281 - val_loss: 0.5611 - val_accuracy: 0.7300
Epoch 2/2
375/375 [==============================] - 182s 486ms/step - loss: 0.5394 - accuracy: 0.7385 - val_loss: 0.5513 - val_accuracy: 0.7340
```

if the model overfitted, you would see the opposite trend (training loss < validation loss).

If The validation accuracy is greater than training accuracy. This means that the model has generalized fine. It can be considered in two ways:

- training data had several arduous cases to learn
- validation data containing easier cases to predict

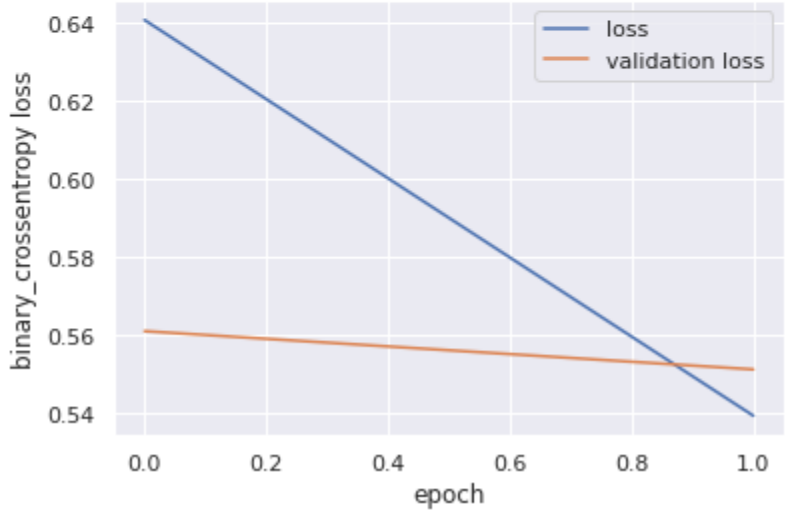
```
[ ]  1 model.evaluate(pad_test, y_test) # pad_text is the padded x_test (consisting of lyrics) and y_test is the valence label
```

```
94/94 [==============================] - 8s 87ms/step - loss: 0.5513 - accuracy: 0.7340
[0.5513045191764832, 0.734000027179718]
```

# LSTM Model Evaluation

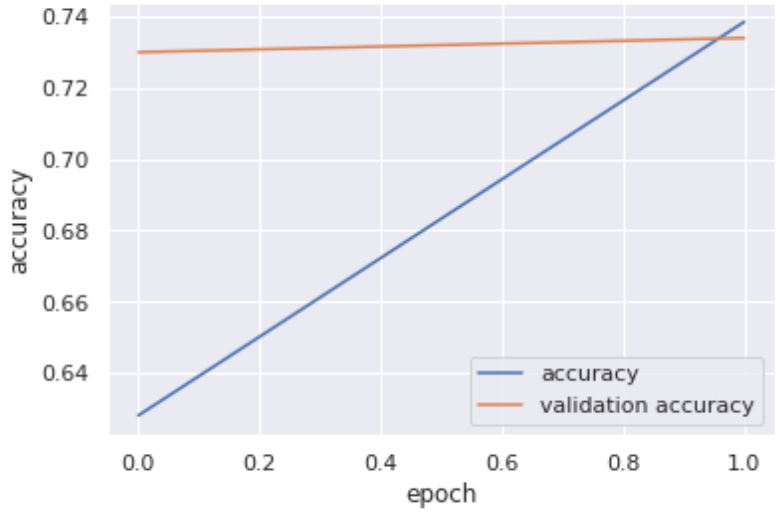
```
[ ]  1 # Evaluating the model
     2 # plotting the loss and validation loss of the model
     3 plt.plot(r.history['loss'], label='loss')
     4 plt.plot(r.history['val_loss'], label = 'validation loss')
     5 plt.xlabel("epoch")
     6 plt.ylabel("binary_crossentropy loss")
     7 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb3428eec50>
```



```
[ ]  1 # plotting the accuracy and validation accuracy of the model
     2 plt.plot(r.history['accuracy'], label= 'accuracy')
     3 plt.plot(r.history['val_accuracy'], label='validation accuracy')
     4 plt.xlabel("epoch")
     5 plt.ylabel("accuracy")
     6 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb3428e1a90>
```




```
[ ]  1 # extract the predicted probabilities
     2 p_pred = model.predict(pad_test)
     3 p_pred = p_pred.flatten()
     4 print(p_pred.round(2))
```

```
[0.82 0.18 0.87 ... 0.1  0.55 0.8 ]
```

```
[ ]  1 y_pred = np.where(p_pred > 0.5, 1, 0)
     2 print(y_pred)
```

```
[1 0 1 ... 0 1 1]
```

```
[ ]  1 from sklearn.metrics import confusion_matrix
     2 print(confusion_matrix(y_test, y_pred)) # confusion matrix on testing data
```

```
[[1146  370]
 [ 428 1056]]
```

```
[ ]   1 from sklearn.metrics import classification_report
      2 print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.73      0.76      0.74      1516
               1       0.74      0.71      0.73      1484

        accuracy                           0.73      3000
       macro avg       0.73      0.73      0.73      3000
    weighted avg       0.73      0.73      0.73      3000
```

We can use confusion matric to calculate precision, recall, accuaracy, score, sensitivity and specificity.

Use the confusion matrix to calculate:

TP (True Postive) TN (True Negative) FN (False Negative) FP (False Positve) Use that information to calculate precision and recall. Then use precision and recall to calculate F1-score.

Precision, recall and F1-score can also be calculated using sklearn library.

```
True_Negative = 1146
False_Positive = 370
False_Negative = 428
True_Positive = 1056
```

```
recall = True_Positive/(True_Positive + False_Negative)
recall
Recall = "{:.2f}".format(recall)
print(Recall)
```

```
0.71
```

```
precision = True_Positive/(True_Positive + False_Positive)
precision
Precision = "{:.2f}".format(precision)

print(Precision)
```

```
0.74
```

```
F1_score = (precision * recall / (precision + recall)) * 2
F1_score
f1_score = "{:.2f}".format(F1_score)

print(f1_score)
```

```
0.73
```

```
# accuracy = (True_Positive + True_Negative) /(3000) # number of
# (lyrics documents) rows in testing dataset: 3000 in 80: 20 split
accuracy = (True_Positive + True_Negative) /(True_Positive + True_Negative + False_Negative + False_Positive)
accuracy

Accuracy = "{:.2f}".format(accuracy)

print(Accuracy)
```

```
0.73
```

```
# Sensitivity is a measure of the proportion of actual positive cases
# that got predicted as positive (or true positive).
# Sensitivity is also termed as Recall.

sensitivity =  (True_Positive)/(True_Positive + False_Negative)
sensitivity
Sensitivity = "{:.2f}".format(sensitivity)

print(Sensitivity)
```

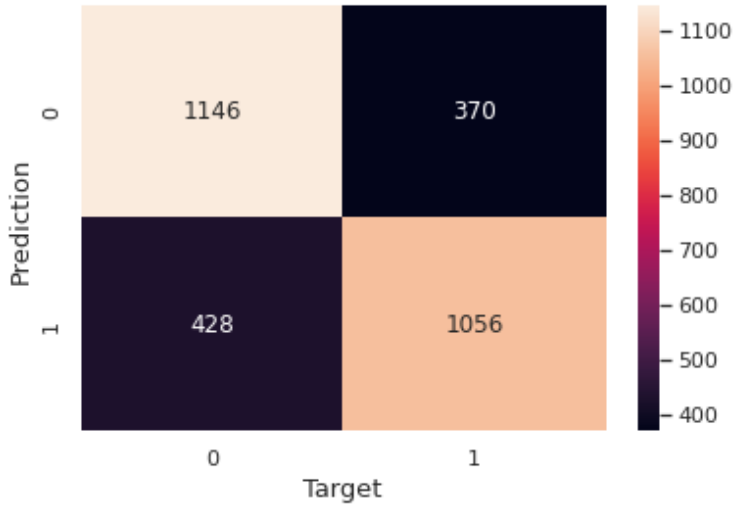
```
0.71
```

```
specificity = (True_Negative)/(True_Negative + False_Positive)
specificity
Specificity = "{:.2f}".format(specificity)

print(Specificity)
```

```
0.76
```

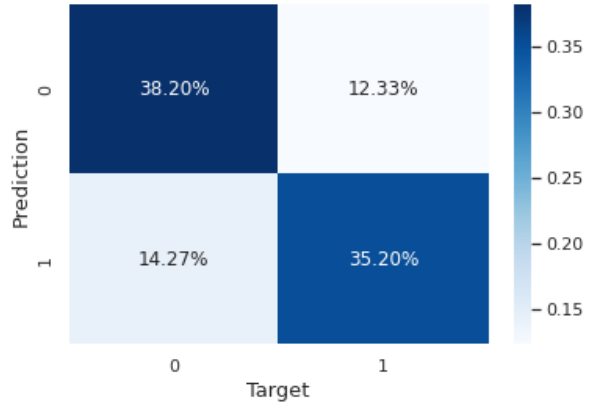
```
import seaborn as sns
q = sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='g')
# fmt='g' to show numbers in decimal notation instead of E-notation.
# annot=True to show 0 and 1 in the heatmap.

q.set_xlabel("Target", fontsize = 13)
q.set_ylabel("Prediction", fontsize = 13)
```

```
Text(30.5, 0.5, 'Prediction')
```



```
# To see what percentage of my data is represented in each quadrant
p = sns.heatmap(confusion_matrix(y_test, y_pred)/np.sum(confusion_matrix(y_test, y_pred)),
                annot=True, fmt='.2%', cmap='Blues')

p.set_xlabel("Target", fontsize = 13)
p.set_ylabel("Prediction", fontsize = 13)
```

```
Text(30.5, 0.5, 'Prediction')
```

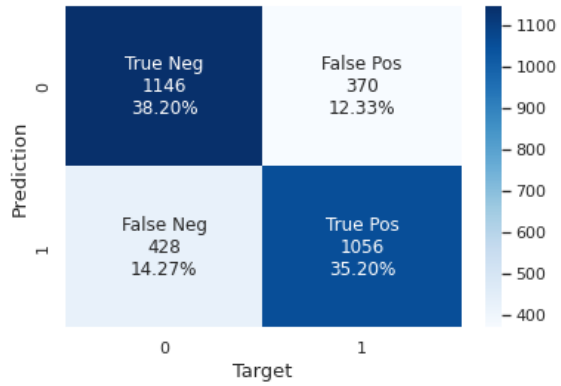



```
[ ]  #Get the confusion matrix
     cf_matrix = confusion_matrix(y_test, y_pred)
     print(cf_matrix)
```

```
[[1146  370]
 [ 428 1056]]
```

```
[ ]  group_names = ['True Neg','False Pos','False Neg','True Pos']
     group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
     group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]
     labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
     labels = np.asarray(labels).reshape(2,2)
     c = sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Blues')
     c.set_xlabel("Target", fontsize = 13)
     c.set_ylabel("Prediction", fontsize = 13)
```

```
Text(30.5, 0.5, 'Prediction')
```



## Predicting the sentiment of any unknown text

```python
def predict_sentiment(text):
  # preprocessing the given text
  text_seq = tokenizer.texts_to_sequences(text)
  text_pad = pad_sequences(text_seq, maxlen=padded_seq_len)
  # Pad the sequence so that it same size as train sequence length (Vocab_size)

  # predicting the class
  predicted_sentiment = model.predict(text_pad).round()

  if predicted_sentiment == 1.0:
    return(print('Positive sentiment'))
  else:
    return(print('Negative sentiment'))
```

```python
# I Hate U, I Love U Song by Gnash --> arouses sad feelings and model predicted it right

text = ["feeling used but I am still missing you And I can't see the end of this Just wanna feel your kiss A
predict_sentiment(text)
```

```
Negative sentiment
```

```python
# Heaven by Beyonce -> sad song

text = ["I fought for you The hardest, it made me the strongest So tell me your secrets I just can't stand t
predict_sentiment(text)
```

```
Negative sentiment
```

```python
# Party in the USA by Miley Cyrus -> Happy song
text = ["I hopped off the plane at LAX with a dream and my cardigan Welcome to the land of fame, excess, who
predict_sentiment(text)
```

```
Positive sentiment
```

```python
# Ariana Grande 7 rings -> arouses positive feelings predicted correctly
text = ["Yeah, breakfast at Tiffany's and bottles of bubbles Girls with tattoos who like getting in trouble
predict_sentiment(text)
```

```
Positive sentiment
```

```python
# Ariana Grande Thank you next next
text = ["Thought I'd end up with Sean But he wasn't a match Wrote some songs about Ricky Now I listen and la
predict_sentiment(text)
```

```
Positive sentiment
```

```
# Dark Paradise by Lana Del ray -> sad song predicted correctly as Negative
text = ["""All my friends tell me I should move on
I'm lying in the ocean, singing your song
Ahh
That's how you sang it
Loving you forever can't be wrong
Even though you're not here, won't move on
Ahh
That's how we played it
And there's no remedy for memory, your face is like a melody
It won't leave my head
Your soul is haunting me and telling me that everything is fine
But I wish I was dead
Every time I close my eyes, it's like a dark paradise
No one compares to you
I'm scared that you won't be waiting on the other side
Every time I close my eyes, it's like a dark paradise
No one compares to you
I'm scared that you won't be waiting on the other side
All my friends ask me why I stay strong
Tell 'em when you find true love, it lives on
Ahh
That's why I stay here
And there's no remedy for memory, your face is like a melody
It won't leave my head
Your soul is haunting me and telling me that everything is fine
But I wish I was dead
Every time I close my eyes, it's like a dark paradise
No one compares to you
I'm scared that you won't be waiting on the other side
Every time I close my eyes, it's like a dark paradise
No one compares to you
But there's no you, except in my dreams tonight

I don't want to wake up from this tonight
Oh-oh-oh-oh-hah-hah-hah-hah
I don't want to wake up from this tonight
There's no relief, I see you in my sleep
And everybody's rushing me, but I can feel you touching me
There's no release, I feel you in my dreams
Telling me I'm fine
"""]
predict_sentiment(text)
```

```
Negative sentiment
```

## Save model

Reference: https://keras.io/api/models/

```
[ ]
from keras.models import model_from_json
# serialize model to json
json_model = model.to_json()
#save the model architecture to JSON file
with open('/content/drive/My Drive/Colab Notebooks/data/models_saved/lyrics_LSTM_model_73.json', 'w') as json_file:
    json_file.write(json_model)

#saving the weights of the model
model.save_weights('/content/drive/My Drive/Colab Notebooks/data/models_saved/lyrics_LSTM_weights_73.h5')
#Model loss and accuracy
loss,acc = model.evaluate(pad_test, y_test, verbose=2)
```

```
94/94 - 12s - loss: 0.5513 - accuracy: 0.7340 - 12s/epoch - 124ms/step
```

## Load model

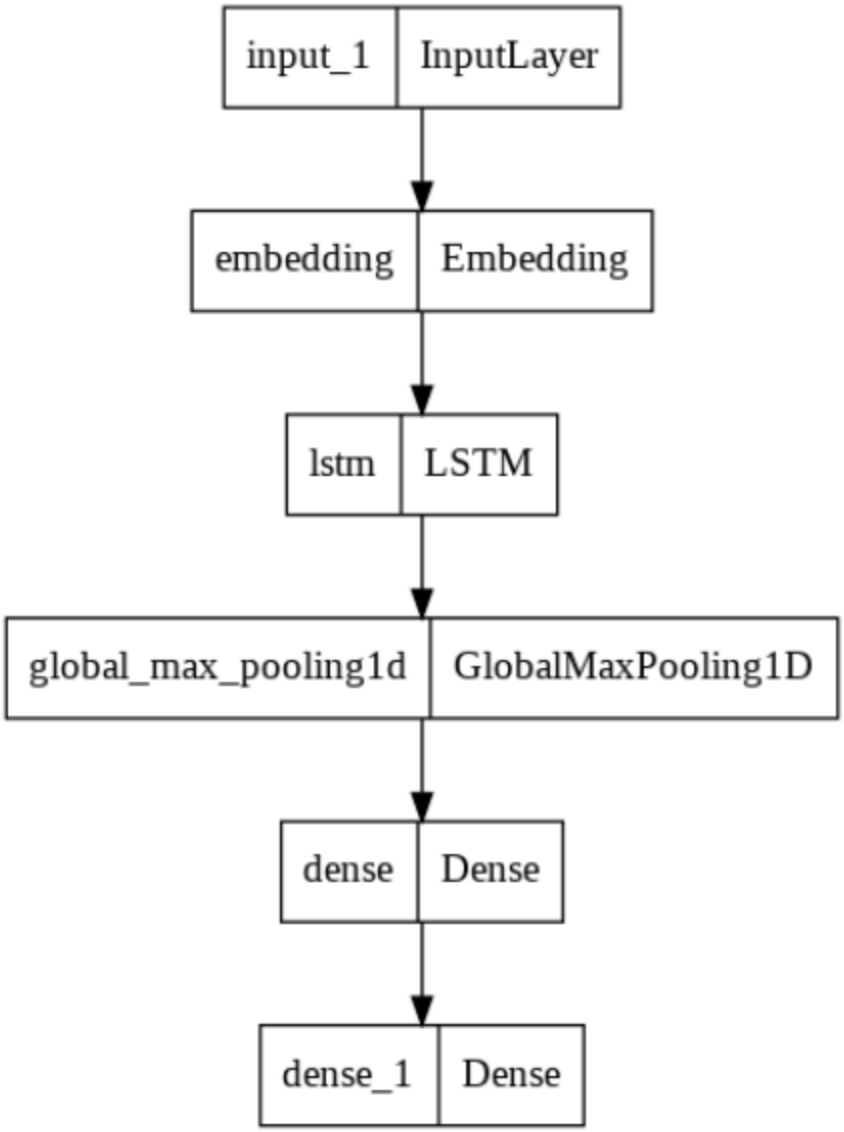
```
[ ]
from keras.initializers import glorot_uniform
#Reading the model from JSON file
with open('/content/drive/My Drive/Colab Notebooks/data/models_saved/lyrics_LSTM_model_73.json', 'r') as json_file:
    json_savedModel= json_file.read()


#load the model architecture
model_j = tf.keras.models.model_from_json(json_savedModel)
model_j.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1286)]            0

 embedding (Embedding)       (None, 1286, 20)          601160

 lstm (LSTM)                 (None, 1286, 15)          2160

 global_max_pooling1d (Globa  (None, 15)               0
 lMaxPooling1D)

 dense (Dense)               (None, 32)                512

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 603,865
Trainable params: 603,865
Non-trainable params: 0
```

```
[ ]
model_j.load_weights('/content/drive/My Drive/Colab Notebooks/data/models_saved/lyrics_LSTM_weights_73.h5')
```

```
[ ]
#Compiling the model
model_j.compile(loss='binary_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

```
[ ]  1 from tensorflow.keras.utils import plot_model
     2 plot_model(model_j)
```



```python
# Predicting the sentiment of any lyrics text

def predict_sentiment1(text):
  # preprocessing the given text
  padded_seq_len = 1286  # derived above
  #padded_seq_len = 1722 # change to 1722 if you get 1722 above.
  text_seq = tokenizer.texts_to_sequences(text)
  text_pad = pad_sequences(text_seq, maxlen=padded_seq_len)
  # Pad the sequence so that it same size as train sequence length (Vocab_size)

  # predicting the class
  predicted_sentiment = model_j.predict(text_pad).round()

  if predicted_sentiment == 1.0:
    return(print('Positive sentiment'))
  else:
    return(print('Negative sentiment'))
```

```python
# Party in the USA by Miley Cyrus -> Happy song
text = ["I hopped off the plane at LAX with a dream and my cardigan Welcome to the land of fame, excess, whoa am I gotta fit i
predict_sentiment(text)
```

```
Positive sentiment
```

```python
# I Hate U, I Love U Song by Gnash --> arouses sad feelings and model predicted it right

text = ["feeling used but I am still missing you And I can't see the end of this Just wanna feel your kiss Against my lips And
predict_sentiment(text)
```

```
Negative sentiment
```

```python
# Heaven by Beyonce -> sad song

text = ["I fought for you The hardest, it made me the strongest So tell me your secrets I just can't stand to see you leaving
predict_sentiment(text)
```

```
Negative sentiment
```

```python
# Ariana Grande 7 rings -> arouses positive feelings predicted correctly
text = ["Yeah, breakfast at Tiffany's and bottles of bubbles Girls with tattoos who like getting in trouble Lashes and diamond
predict_sentiment(text)
```

```
Positive sentiment
```

========== END OF THE REPORT ==========