MOOD CHORDS

(Web-based Music Mood and Genre Classifier)

Ashlesha Shinde*, Chhavi Sharma*, Prahasan Gadugu* &
Supriya Ayalur Balasubramanian*

CONTENTS

* *Master of Science in Data Science, School of Informatics and Computing, Indiana University Bloomington, USA*

ABSTRACT

In this project, we have investigated the performance of various machine learning algorithms and models to classify songs based on their mood and genres using only the lyrics features. We have deployed the final model on a web application using flask and implemented Dynamic learning to retrain our model and improve accuracy.

1 INTRODUCTION

Life takes us through a roller-coaster of emotions. Music is a universal language that allows us to cope, understand, and celebrate these emotions in beautiful ways.[1] Humans organize music based on how they feel after listening to a song. If there was a way to generate a set of songs that was perfectly synchronized with your mood, it would make the process of finding music to compliment your current emotions much more seamless. Song classification on the basis of mood can be a tedious task since it is perspective-based and therefore, different people can have a different emotional reaction to the same song. In such a scenario, the lyrics of a song contain crucial information vital to the reception of the song.

Song lyrics could be organized in several structures like chorus, verse and rhymes. It is easier to identify these structures through music lyrics rather than through their audio. Moreover, songs are generally classified using audio features like rhythm, timbre and harmony or the artist's genre rather than the mood or genre communicated by the song. In this study, we attempt to investigate the performances of musical genre and mood classification using only the Natural Language Processing based features extracted from lyrical data (textual information) of a song.

2 RELATED WORK*

A combination of lyrics and audio features have been in practice since many years for extracting information for Mood and Genre classification. One of the earliest studies by Yang and Lee combined lyrics and audio data for Music Information Retrieval. They combined lyric Bag-of-Words(BOW) features and the 182 psychological features proposed in the General Inquirer to render mood categories that audio-based classifiers could not. This improved the accuracy of the classification algorithm by 2.1

** [1] Moodtape: Web Application Using Spotify API to Create Mood-Generated Playlists*
By- Mohith Subbaro, July 21, 2018

Several lyrics recognition system using a large vocabulary continuous speech recognition (LVCSR) technique based on HMM (hidden Markov model), acoustic model and a tri-gram language model were developed to retrieve information about music from lyrical data.

A semantic and structural analysis of song lyrics focusing on structure detection, semantic search and theme-based classification has been performed in [2]. They collected data from several web sources and performed sequence matching on it to detect typing errors, or parts of the text that is not part of the original lyrics.

Cyril et al. conducted a research emphasizing the importance of lyrics in predicting the emotion or mood of music. This established a strong correlation between audio data and textual lyrical data of a song.

X. Hu and Downie performed their study based on lyrics text and used its features to classify 18 mood categories derived from user tags. Their recent research reiterated the strong relation between lyrical and audio data and its combination being helpful towards predicting the mood of a song.

Another old study on genre classification in lyrics conducted by Neumayer and Rauber. They combined lyric and audio features of the data and used them to classify the genre of a song. Lyric analysis included rhyme pattern, part of speech characteristic and text statistic features of a song. They used Bag of Words for feature extraction and Term Frequency-Inverse Document Frequency to assign weights to the features. They selected features by eliminating term occurring ver frequently or very rarely in the text since such terms do not contribute much towards the prediction.

Now that the correlation between lyrical data and efficient music mood and genre classification has been established, we performed our experiments only using textual lyric data to predict both mood and genre of a song using Natural Language Processing.

** [2] Rudolf Mayer and Andreas Rauber, 2010, "Multimodal Aspect of Music Retrieval: Audio, Song Lyrics – and Beyond?"*
Z.W.Raś and A.A. Wieczorkowska
(Eds.): Adv. In Music Inform. Retrieval, SCI 274, pp. 333-363
** Related work referred from Genre and Mood Classification Using Lyric Features by Teh Chao Ying, Shyamala Doraisamy, Lili Nurliyana Abdullah*
Faculty of Computer Science and Information Technology

3 PRODUCT FEATURES

Our basic idea is to prototype a web-based application that scrapes lyrical data from the web and uses it to extract important features to classify the songs on the basis of 2 categories:

1. **Mood Classification:**
   We will be using randomly sub-sampled and pre-labeled data by Sebastian Raschka to classify the songs into 2 moods:
   Happy or Sad
2. **Genre Classification:**
   We will be using a Kaggle dataset collected from MetroLyrics to classify the songs into 8 genres:
   Hip-Hop, Metal, Electronic, Pop, Country, Rock , Folk Indie R& B, Jazz

After training and modeling the data for predictions, we will be deploying our best models for Mood Classification and Genre Classification on a Flask-based web-application. This application will incorporate **Dynamic Learning** using which it will allow human-computer interaction by taking feedback of the user on the predicted label and re-training the data accordingly with correct predictions.

4 DATA SET DESCRIPTION AND DATA PRE-PROCESSING

As mentioned above, we used different datasets for Mood and Genre classifications.

4.1 Mood Classification

We are using the randomly subsampled and labelled data by Sebastian Raschka ("Python Machine Learning" Co-Author). The data has been pre-processed by him in the following manner:

1. A 10,000-song subset was downloaded from the Million Song Dataset.

2. Lyrics were automatically downloaded from LyricWikia and all songs for which lyrics have not been available were removed from the dataset.
3. An English language filter was applied to detect and remove all non-English songs.
4. The remaining songs were randomly subsampled into a 1000-song training dataset and 200-song validation dataset.

4.2 Genre Classification

Initially, we took a Kaggle dataset from Kaggle which was collected from MetroLyrics for genre classification. The data was found to have the following classes: Country, Electronic, Folk, Hip hop, Indie, Jazz, Metal, Pop, R& B and Rock(with not available and others). However, the data was highly imbalanced[Figure 1] with Rock class data being the most in the dataset. Thus, for cleaning the data, we removed the unavailable lyric information and then balanced the data set by under sampling the Rock genre[Figure 2]. We also clubbed a few genres due to their similarities such as Folk, Indie and R& B and then over-sampled the information in order to make sure we get a balanced information. The final data set comprises 20,000 song lyrics with 8 labels [Hip-Hop, Metal, Electronic, Pop, Country, Rock , Folk Indie R& B, Jazz). Moreover, we split the data into train set with 80% data and validation set with 20% data in order to train and evaluate our models.

4.3 Test Environment

We are webscraping the lyrics from Musix match website for testing how our models work in the production environment (for both mood and genre classifications) which will be further discussed with the flask model.

5 APPROACH

On a broad scale, we basically have two text classification problems: Mood classification (binary classification) and Genre Classification (multi-class classification)[Figure 22]. Thus, while studying different approaches employed in text classification, we came across **Google Text Classification Algorithm**[3] which was generated by Google developers after extensive research and practical modeling on various textual datasets. It provides the following workflow for any text classification problem:

1. Step 1: Gather Data
2. Step 2(i): Explore Your Data
3. Step 2(ii): Choose a Model
4. Step 3: Prepare Your Data
5. Step 4: Build, Train, and Evaluate Your Model
6. Step 5: Tune Hyperparameters
7. Step 6: Deploy Your Model

5.1 Exploratory Data Analysis

Since it is a text classification problem, the data visualization of most common words in the happy and sad mood can be given by the word cloud representation. As we can see in the figure[Refer Figures 11-15], we can see that for Mood classification dataset, "got", "baby","man", "love" are the most common for 'happy' mood and "tell","life","nigga" are most common words for 'sad' mood predictions. For genre classification, we have taken the top three genres into consideration to know the most common words in the top genre. Pop genre has words "cause", "know","love" as the most common words whereas Metal has "never", "mind", "let". Rock has "life", "time", "love" as the most common words.

* *[3] Google developers Text Classification Guide*
`https://developers.google.com/machine-learning/guides/text-classification/`
Creative Commons Attribution 3.0 License

5.2 Gather Data

Data-collection is the most integral part of any text classification problem since a text classifier is completely dependent on the goodness of the data it is built from. Data can be collected or leverage from public APIs or pre-built datasets in GitHub or Kaggle. We already discussed where we gathered the data from for Mood and Genre Classification.

5.3 Explore Your Data

It is crucial to explore the data and understand its characteristics before proceeding with modeling. This step will provide you better insights into the data which may further help in improving the model and saving computational resources by predicting better with lesser data. Some key metrics to be collected are:

1. Number of samples
2. Number of classes
3. Number of samples per class
4. Number of words per sample
5. Frequency distribution of words
6. Distribution of sample length

5.4 Choose a Model

In this step, we use the key metrics obtained from the data in data exploration to choose the classification model to be used from a large set of modeling options available.

They designed a simple algorithm for Data Preparation and Data Modeling:

1. The number of samples/number of words per sample ratio is calculated.

2. If this ratio is less than 1500, tokenize the text as n-grams and use simple models like Support Vector Machines, Gradient-Boosted Decision Trees or Multi-Layer perceptron (MLP) model for classification classify them (left branch in [Figure 22]):
   a) Split the samples into word n-grams and convert these n-grams into vectors.
   b) Score the importance of the vectors and then select the most important (generally around top 20,000 features) using the scores. This can be done by using f_ classif or chi2 scores.
   c) Build the models.
3. If the ratio is greater than 1500, tokenize the text as sequences and use models like Recurrent Neural Network (RNN), stacked RNN, Convolutional Neural Networks (CNN), CNN-RNN and Separable Convolutional Networks (sepCNN) model to classify them (right branch in [Figure 22]):
   a) Split the samples into words and select the top 20K words based on their frequency.
   b) Convert the samples into word sequence vectors.
   c) If the original number of samples/number of words per sample ratio is less than 15K, using a fine-tuned pre-trained embedding with the sepCNN model will probably provide the best results.
4. Measure the model performance with different hyperparameter values to find the best model configuration for the dataset.
5. Deploy the best model.

Thus, to choose the model, we need to calculate the number of samples/number of words per sample to know whether we need to follow the n-gram model or the sequence model. The difference between both the models is that n-gram model does not take the ordering of words into account. However, sequence models account for word ordering using word-embedding and word vectors and therefore, should be used when the order of words is important.

5.5 Prepare your Data

After choosing the model, we need to first, prepare our data into a format that can be recognized by a model. Any model takes and understands numerical data as input. Therefore, we need to convert text data into numerical vectors. For this, we first tokenize the text.

5.5.1 *Tokenization*

In tokenization, text is divided into smaller units like words or sub-texts to form the vocabulary of the dataset. For N-gram vectors model, as per their research, unigrams and bigrams provide the best accuracy and consumes lesser computation time. For Sequence model, we tokenize the text into a sequence of characters or words. However, as per their study, unless and until, there are a lot of typos in the data, word-level representation gives better results.

5.5.2 *Feature Selection*

After tokenization, the data may take the form of several tokens, however each of them will not be required. Thus, some tokens can be dropped and only most important tokens (contributing the most towards prediction) can be retained as features. The most informative features can be extracted by statistical functions like **f_ classif** or **chi2**. As per Google research, generally the accuracy peaks around top 20,000 features. Skipping this step could lead to higher computation time or overfitting. For sequence models, we convert the tokens into sequence vectors and pad them to a fixed-length sequence.

5.5.3 *Vectorization*

Vectorization takes care of converting textual data into numerical data that can be understood by the modeling algorithm. Vectorization techniques are different for different tokens:
For N-gram model, the n-grams need to be converted into numerical vectors. We assign indexes to n-grams and then use one of the following techniques:

1. **One-hot encoding:** Every sample text in the data is represented as a vector indicating the presence or absence of a token in the text.

2. **Count encoding:** Every sample text is represented as a vector indicating the count of a token in the text.

3. **Tf-Idf encoding:** It serves as an improvement on the previous two approaches since it penalizes the words that occur in similar frequencies across all the documents since in that case they do not contribute towards predicting different labels. As per research, Tf-Idf performs marginally (around 0.25-15% higher) than other two. However, it occupies more memory and is computationally more expensive.

For sequence model, the sequences need to be converted into numerical vectors. Vectorization of token sequences can be done in two ways:

1. **One-hot encoding:** Sequences are represented using word vectors in n-dimensional space where n is the size of vocabulary. For character tokens and small vocabulary, it is a better approach. For word vectors or larger vocabulary, the one-hot vectors will be very sparse and thus, inefficient.

2. **Word embeddings:** This takes the semantic of the language into account considering that words have some meaning. Therefore, it forms a dense vector representation [Figure 23] where location and distance between words indicates how similar they are semantically. This embedding layer generally forms the first layer of the sequence model.

5.5.4 *Label Vectorization*

After normalizing the entire data, we also need to preprocess the labels. This involves converting labels into values in range [0, number of classes-1]. Internally, the network uses one-hot vectors to represent this. The representation depends on loss-function and activation function in the last layer.

5.6 Build, Train, and Evaluate Your Model

As per the chosen n-gram or sequence model, the algorithm can be chosen for target prediction. Algorithms performing better for N-gram model:

1. Support Vector Model

2. Gradient-boosted Decision Trees
3. Multi-layer perceptron

Algorithms performing better for sequence model:

1. Recurrent Neural Network (RNN)
2. stacked RNN
3. Convolutional Neural Network (CNN)
4. CNN-RNN
5. CNN
6. Separable CNN (sepCNN)

5.7 Hyperparameter Tuning

For drfining and training the model, there are a number of hyperparameters for each model that must be tuned to a value giving the best accuracy. For neural networks, the most important parameters are:

1. Number of layers in the model
2. Number of units per layer
3. Dropout rate
4. Learning rate

5.8 Deploy your Model

In the end, after choosing and hyperparameter tuning for the best model and obtaining the best accuracy, the model can be deployed in various forms and platforms like GoogleCloud, as a web-application using Flask etc.

6 EXPERIMENTATION AND RESULTS

We began by following the Google Text Classification approach and calculated the number of samples/number of words per sample ratio(S/W ratio) for booth Mood Classification and Genre Classification data sets.
S/W ratio for Mood classification: 10.20
S/W ratio for Genre classification: 203.79

Thus, both follow N-gram Model approach. However, we did deviate a little from the Google Classification algorithm to experiment more and verify if it works well. Since we are mainly focused on text based prediction rather than acoustic based prediction, we wanted to focus more on text feature extraction:
The typical feature extraction methods followed in N-gram model are, Bag of Words, TF-IDF which means Term-Frequency Inverse Document Frequency and N-grams. We wanted to go a step further from Google Text classification algorithm and therefore, implemented the GloVe and word embeddings (as a layer in the Neural Networks):

1. **Bag Of Words:**
   This is a text mining technique that builds a predefined set of words and calculates the word count for each word. This word count matrix is then used to extract different feature representations. We have a huge amount of data owing to which the number of words in the lyrics text will be huge. Therefore, in order to get better results, we will consider the 20,000 topmost frequent words as our feature set and train our models on these features.

2. **TF-IDF:**
   Tf-Idf (Term Frequency Inverse Document Frequency) is a text mining technique used to categorize documents. It emphasizes on words that occur frequently in a particular song lyric, while at the same time de-emphasizing words that occur frequently in song lyrics. This method will help us concentrate on portions which might be less frequent but have more significance for the overall prediction of the song's mood and genre. To limit the number of features getting created, we selected

the 20,000 topmost features as specified in the Google Text classification algorithm.

3. **N-grams:**

   We used N-grams to create unigrams and bigrams. This method provides more contextual information on the song lyrics since it takes the order a little bit into account. It combines n sequential words.

4. **GloVe:**

   GloVe is basically a pre-trained word embedding using a large multiple corpora mostly collected from the Wikipedia data. It is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.[4] The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. The tools provided in this package automate the collection and preparation of co-occurrence statistics for input into the model. The core training code is separated from these preprocessing steps and can be executed independently.

6.1 Mood Classification:

This is a binary classification problem wherein based on the lyric, we need to predict whether the song is Happy/Sad. Hence, we have two target labels. The train set has 1000 songs and validation set has 200 songs.

6.1.1 *Baseline Modeling Mood Classification:*

To classify the lyrics into one of the two classes : Happy/Sad, we started with testing several baseline binary classification algorithms to find out which

** [4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation* model is working better. The performance evaluation is done on the validation set. We implemented the baseline models using the Scikit-Learn library in Python.[Figure 9]

1. **Logistic Regression:**
   It is a predictive classifier in which there are one or more variables determining the outcome. The probabilities describing the possible outcomes of a single trial follow a sigmoid curve. Logistic Regression model with L2 regularization has been applied to take care of overfitting which gives the least accuracy of 49.50 percent.

2. **Naïve Bayes:**
   The Naïve Bayes algorithm assumes that every feature, i.e, each word or a group of words is conditionally independent of each other. The Multinomial Naïve Bayes approach has given an accuracy of 58 percent.

3. **Random Forests:**
   A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. We set the parameters of the classifier to have 100 trees in the forest and the maximum depth of the tree as 10 and achieved an accuracy of 54 percent.

4. **Linear SVC:**
   It is a supervised learning algorithm which designates a hyperplane in an n-dimensional space where each data instance is plotted that efficiently differentiates each class. We used LinearSVC that finds the best fit hyperplane to categorize the data. Thereafter, the features can be fed into the model to predict the class it belongs to. This classifier has given 54 percent accuracy.

The summary of the performance of each baseline classifiers has been given in the table appendix quotation[Figure 20].

Also, since this project deals with mood classification analysis of the lyrical data, we realized that bag of words concentrates more on word count rather than the logical construct of data. It is quite possible that a word may occur less frequently but is clearly indicative of a class. To overcome this shortcoming, we used TF-IDF model for feature extraction in the top 3 baseline classifiers for better accuracy.

6.1.2 *Improvement on Baseline Models for Mood Classification:*

After looking at the accuracies of the baseline models, we wanted to analyze if the accuracy will improve by applying pipeline (sequence of list of transforms with a final estimator) and tweaking the Term Frequency – Inverse Document Frequency (TF-IDF) feature extraction process i.e, tuning the hyperparameters of the baseline models. We selected the top three best performing models and implemented the algorithms with the tuned parameters to see if there is any significant improvement in the accuracy.[Figure 10]
The summary of the performance of each conventional classifiers along with TF-IDF feature extraction has been given in the table appendix quotation.[Figure 21]

6.2 Genre Classification:

This is a multi-class classification problem wherein based on the lyrics, we need to predict the genre of the song. Here, we have 8 target labels i.e, **Hip-Hop, Metal, Electronic, Pop, Country, Rock , Folk Indie R& B, Jazz**. The dataset has 20000 lyrics and data is split into 80% i.e, 16000 songs as train set and 20% (4000 songs) as validation set.

6.2.1 *Baseline Modeling Genre Classification:*

To classify the lyrics into one of the eight genres, we started with testing several baseline classification algorithms to find out which model is working better. The performance evaluation is done on the validation set. We implemented the baseline models using the Scikit-Learn library in Python.[Figure 3]

1. **Logistic Regression:**

   It is a predictive classifier in which there are one or more variables determining the outcome. The probabilities describing the possible outcomes of a single trial follow a sigmoid curve. Logistic Regression model with L2 regularization has been applied to take care of overfitting which gives an accuracy of 34.69 percent.

2. **Naïve Bayes:**
   The Naïve Bayes algorithm assumes that every feature, i.e, each word or a group of words in the lyrics is conditionally independent of each other. The Multinomial Naïve Bayes approach has given the least accuracy of 28.66 percent.

3. **Random Forests:**
   A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. We set the parameters of the classifier to have 100 trees in the forest and the maximum depth of the tree as 10 and achieved an accuracy of 34.57 percent.

4. **Linear SVC:**
   It is a supervised learning algorithm which designates a hyperplane in an n-dimensional space where each data instance is plotted that efficiently differentiates each class. We used LinearSVC that finds the best fit hyperplane to categorize the data. Thereafter, the features can be fed into the model to predict the class it belongs to. This classifier has given 32.99 percent accuracy.

The summary of the performance of each baseline classifiers has been given in the table appendix quotation[Figure 17].

Also, since this project deals with genre classification analysis of the lyrical data, we realized that bag of words concentrates more on word count rather than the logical construct of data. It is quite possible that a word may occur less frequently but is clearly indicative of a class. To overcome this shortcoming, we used TF-IDF model for feature extraction in the top 3 baseline classifiers for better accuracy.

6.2.2 *Improvement on Baseline Models for Genre Classification:*

After looking at the accuracies of the baseline models, we wanted to analyze if the accuracy will improve by applying pipeline (sequence of list of transforms with a final estimator) and tweaking the Term Frequency – Inverse Document Frequency (TF-IDF) feature extraction process i.e, tuning the hyperparameters of the baseline models. We selected the top three best performing models and implemented the algorithms with the tuned parameters to see if there is any significant improvement in the accuracy.[Figure 4] The summary of the performance of each conventional classifiers along with TF-IDF feature extraction has been given in the table appendix quotation.[Figure 18]

6.3 Deep Learning Approach:

Multilayer perceptrons often known as artificial neural networks or just neural networks is composed of more than one perceptron. They comprise an input layer that receives an input signal, an output layer that makes decisions or predictions about the input, and an arbitrary number of hidden layers that are the true computational engines of the MLP.[5] Multilayer perceptrons are often applied to supervised learning problems. They train on a set of input-output pairs and learn to model the correlation between those inputs and outputs.[5]
RNNs make use of sequential information to make predictions. They have a 'memory' which captures information about what has been calculated so far. Traditional neural networks assume that all inputs are independent of each other. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. E.g., if you want to predict the next word in a sentence, you better know which words came before it. RNNs are called recurrent because they perform the same computations on each element of a sequence, with the output being depended on the previous computations.[6] **Long Short Term Memory** networks – usually just called 'LSTMs' – are a special kind of RNN, capable of learning long-term dependencies by remembering information for long periods of time. [6] For

the below mentioned neural networks, we followed the basic tokenization of the lyrics information and then encoded them by post-padding with zeroes. After the vocabulary is built using the tokenization, we initiated the input word embedding layer of every neural network with the size of the vocabulary.

The target labels have been vectorized as follows:

If it is a binary classification problem (in the case of mood classification), binary encoding is done.

If it is a multi-classification problem, (as in Genre classification), we first integer-encode them and then One-hot encode them to generate the binary vectors.

Generally neural network gives best predictions for the classification problems as they have multiple layers through which we can assess the features and activate neurons in further layers where we can tweak to design the model according to the requirements.

6.3.1 *Mood Classification:*

As per the Google Text Classification algorithm, we implemented the Multi-Layer perceptron model, where-in we first vectorized the text data and then generated the feature set using which we built the Input layer and two dense layers with relu activation. As it is a binary classification problem, we used 'sigmoid' in the output layer and 'binary cross entropy' as the loss function. Using these, we obtained an accuracy of 70 percent after 10 epochs with a batch size of 100.

CNN:

Convolution neural networks are generally used in the image classification due to its channel-wise approach to the problems. For CNN as specified by the Yoon Kim's paper, generally, CNN is used for Image Classification owing to its channel-wise approach. However, since this particular dataset presents itself as a text classification problem, single-channel CNN can be

* [5] *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, by Frank Rosenblatt, 1958 (PDF)*
A Logical Calculus of Ideas Immanent in Nervous Activity, W. S. McCulloch & Walter Pitts, 1943
Perceptrons An Introduction to Computational Geometry, by Marvin Minsky & Seymour Papert [6] Blog by DENNY BRITZ - SEPTEMBER 17, 2015
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

used. I used a 2-gram filter window to implement CNN. It generates a single dimensional matrix as an output on which maxpooling 1D can be used to pick the desired values. After converting the output of the CNN into a 1D feature vector (flattening), I added Multi-layer perceptron comprising one layer of 'relu' activation and and other with 'sigmoid' activation as output layer. Since it is a binary classification problem, it has only one output node for which I used 'binary_ crossentropy' as the loss function. Adam optimizer has been used for optimization and 'accuracy' metric has been used for evaluation of the results. This network gave us an accuracy of 69 percent.

CNN with GloVe: Using the same CNN architecture specified above, but with the GloVe Feature extraction, we got an accuracy of 68 percent.[Refer Figure 19]

6.3.2 *Genre Classification:*

CNN:

We implemented a similar CNN approach as we did in Mood Classification but since the Genre is a multi-classification problem with 8 target variables, we used 8 neurons in the output layer with softmax activation and 'categorical cross entropy' . This resulted in an accuracy of 38.13 /

CNN+GloVe Implementation:

We followed a similar CNN approach with GloVe feature extraction which resulted in an accuracy of 40.74 /

RNN(LSTM Implementation):

For RNN, we used the Long-Short-Term-Memory algorithm layer. After an initial Embedding layer (to be used for features extraction from words), we proceeded with the LSTM layer. Thereafter, we implemented a drop-out of 0.2 resulting in 20 percent information loss at Dropout layers. We have additionally added a multi-layer perceptron with a layer of 'relu' activation and other with the 'softmax' activation as an output layer. Categorical cross entropy loss function is used. Adam optimizer has been used for optimization and 'accuracy' metric has been used for evaluation of the results. This gave us an accuracy of 23.47 percent.[Refer Figure 16 for results]

7 'MOOD CHORDS' WEB APPLICATION

7.1 Based Web API

Flask is a light-weight WSGI (Web Server Gateway Interface) web application framework. It takes care of the server-side processing. WSJI is implemented using the Werkzug web application library. Flask began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries. Our code will receive requests and figure out what those requests are dealing with and what they are asking for. It will also designate the response to the user. We have used flask as it makes the process of designing a web environment easier. Flask uses jinja2 for templating. Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure owing to the optional sandboxed template execution environment. We used basic HTML and CSS for designing the User-Interface.

7.2 Website Structure

The home page features a search bar wherein the user is prompted to type in the song title, or artist name, based on which the application will webscrape the lyrics of top 10 songs from Musix Match website. Our application will display the predictions made by the best model. Here, in our case, the best model is Linear SVC classifier for both Mood as well as Genre classifications. Hence, we pickled the classier as an object file and deployed it to our website through Flask. The web application takes the lyrics data song by song and predicts its mood based on Mood Classifier(Linear SVC) and Genre based on the Genre classifier(also Linear SVC). As shown in the figures[Figure 24-27], we display the predictions generated by our model deployed on the website adjacent to the song. The feedback column working and usage is discussed in the Dynamic Learning implementation section. Additionally, we put two more functionalities below the search bar- one, upon clicking would give the predictions of the newly released songs and the other, would give the predictions for top trending songs. These functionalities also use the

Musixmatch website for webscraping the data. Additional website related information is provided in the appendix in the form of screenshots.

8 DYNAMIC LEARNING

Fully supervised feed-forward neural networks cannot cope with situations in which the features of the data keep on changing. Hence, we use Dynamic Learning where the feature set as well as the entire newly formatted information can be dealt without any loss of newly added information. So, basically, a dynamic learning model uses the online information and is also trained online. Data is continuously incorporated into the model through regular updates. Static models are used in an environment where there is a lot of information already available and needs to be tweaked in order to get better models. We lack a new learning environment such as new features of the data etc.,but the dynamic models adapt to the changing information quickly and keep on updating themselves. Here, we have used the Dynamic learning approach so as to keep on adding the new songs that are available. Entertainment industry is providing us with lots of songs so we need to get updated from them and make use of more information in order to make our models predict better.

8.1 Implementation of Dynamic Learning:

In our web-application, we not only build a repository of newly added songs but also learn from the manual labeling of the previously predicted songs. This is implemented[Figure 28] using the feedback column and can be explained as follows: Given a scenario where user queries for the mood of the songs of artist 'Ed Sheeran'. So, we get the top 10 songs by 'Ed Sheeran' and predict their moods. Adjacent to each prediction, we have a column for user feedback. This column confirms from the user whether our prediction for the song is right or wrong. Once the user gives the feedback, we generate a comma-separated file (.csv) file in the back-end which will have the list of the songs that he/she asked for along with the manual labels. If we have multiple feedback for the same song, then we take a max-vote to select the final label. This data is added to our train file and the model is re-trained. Max voting is taken due to the fact that classification of mood of the song is subjective and depends on person to person. As per the figure our dynamic learning environment(code) will take the feedback file from the website and then generate the re-trained best model which will be hosted back on the website.

9 RESULTS AND ANALYSIS

For this project, conventional machine learning algorithms beat the neural network performance by a huge margin. The best models came out to be Linear SVC and Logistic Regression for both Genre and Mood classification. Comparing these models based on Confusion matrices [Figure 5, Figure 6, Figure 7 and Figure 8], on further evaluation, we see that they are predictive individual labels with a good accuracy for mood classification. However, the best models for Genre classification are not good at predicting the individual labels. They predict only Rock, Jazz, Country and Pop genres well.

10 CONCLUSION

In this project, we show the feasibility of predicting mood and genre of a song through song lyrics and text processing techniques. Our comparison of the accuracy for different models on musical genre and mood classification shows that simple models like linear SVC gives much better results as compared to Deep Neural Network techniques like MLP, CNN and RNN. Also, we verified Google Text Classification algorithm to be valid since N-gram model approach algorithm, Support Vector Machines gave us better results than Sequence Models. Also, implementing Dynamic Learning brought human-computer interaction to our project which led to an improved classification accuracy and provided a much-automated learning algorithm.

11 FUTURE SCOPE

In future, we plan to improve our work in the following ways:

1. Work on a larger data collection,
2. Better computation power and memory resources
3. Incorporate audio features like rhythm, harmony and timbre learn better and increase the accuracy.
4. study and implement better machine and trending machine learning techniques in the market and see how the accuracy will be affected.

12 APPENDIX

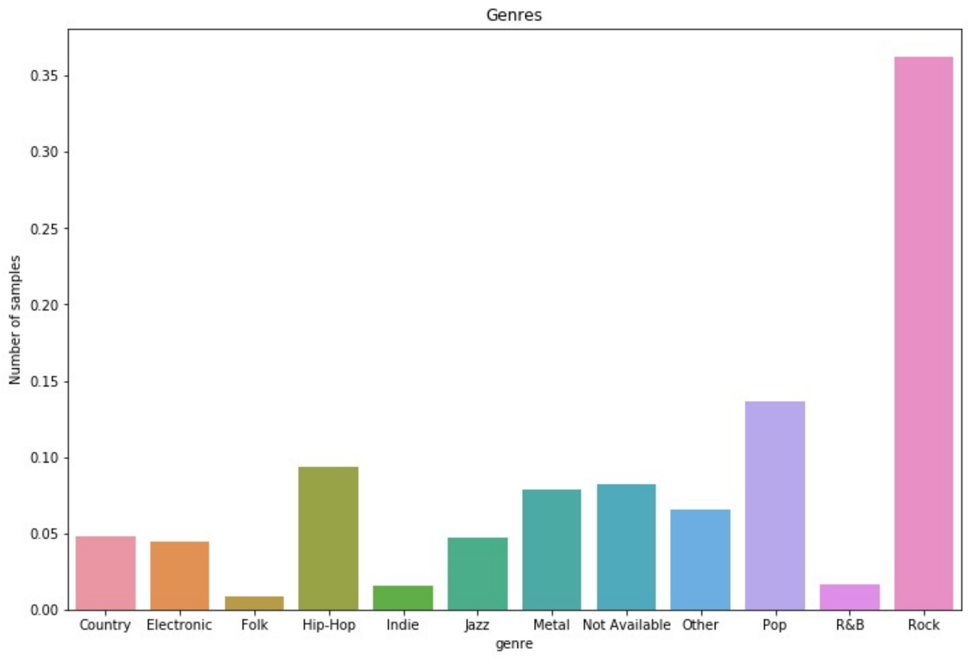


Figure 1: Imbalance in Genre dataset

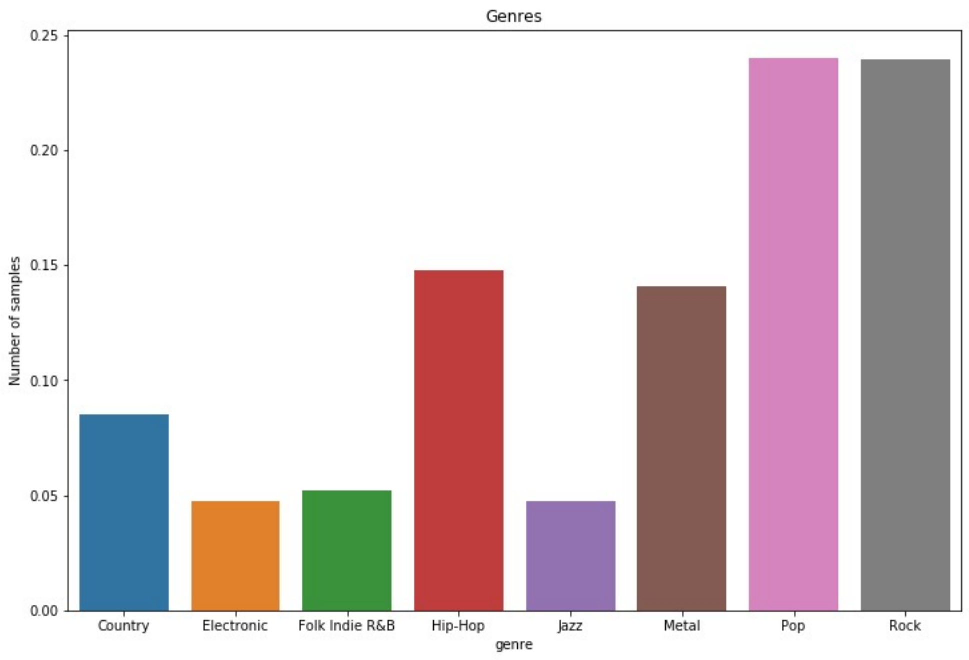


Figure 2: Balanced Genre classes

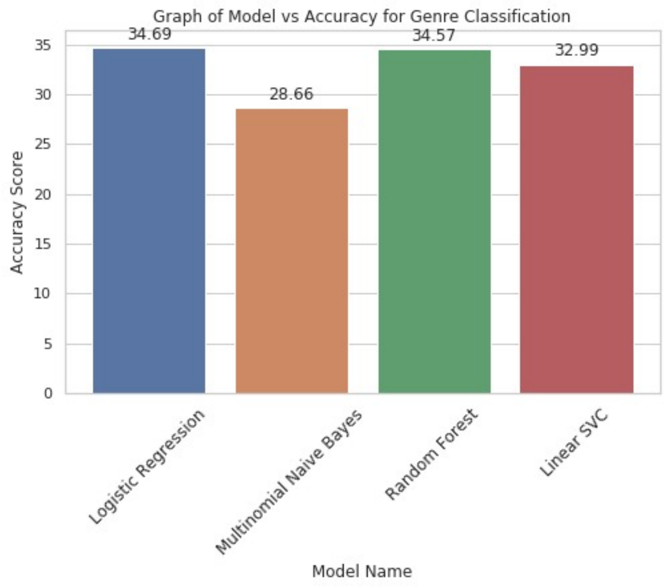


Figure 3: Baseline Model Comparison for Genre Classification

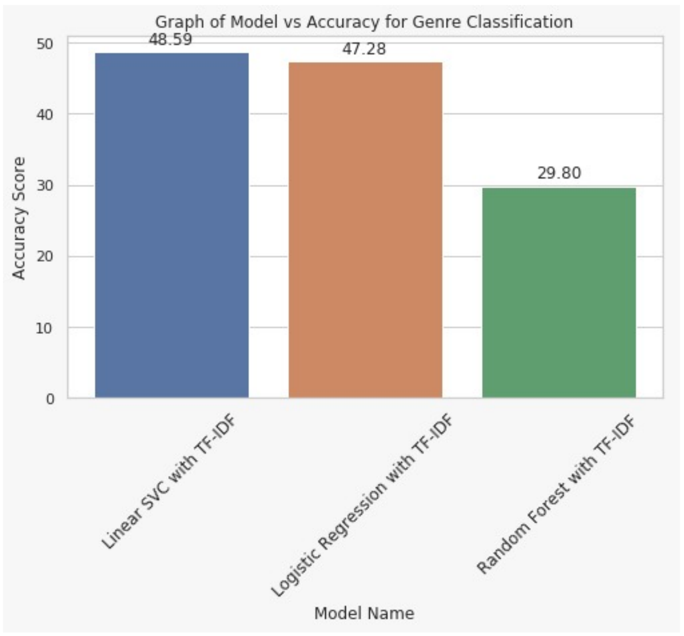
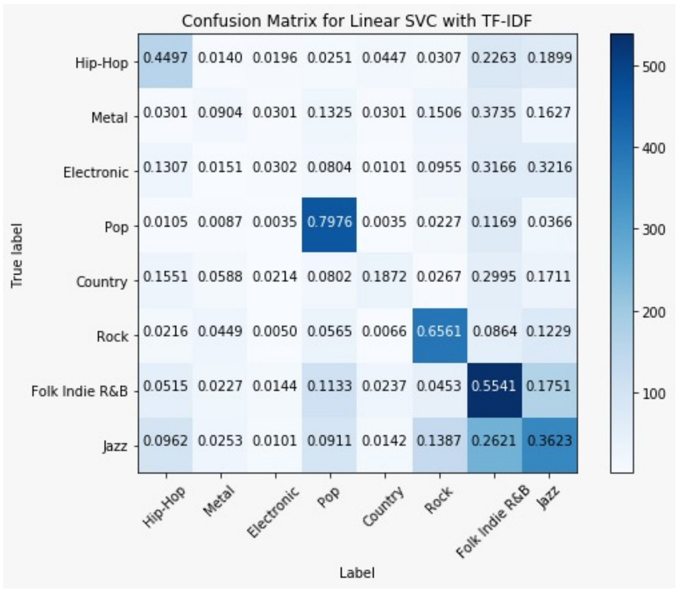


Figure 4: TF-IDF model comparison for Genre Classification



	Hip-Hop	Metal	Electronic	Pop	Country	Rock	Folk Indie R&B	Jazz
Hip-Hop	0.4497	0.0140	0.0196	0.0251	0.0447	0.0307	0.2263	0.1899
Metal	0.0301	0.0904	0.0301	0.1325	0.0301	0.1506	0.3735	0.1627
Electronic	0.1307	0.0151	0.0302	0.0804	0.0101	0.0955	0.3166	0.3216
Pop	0.0105	0.0087	0.0035	0.7976	0.0035	0.0227	0.1169	0.0366
Country	0.1551	0.0588	0.0214	0.0802	0.1872	0.0267	0.2995	0.1711
Rock	0.0216	0.0449	0.0050	0.0565	0.0066	0.6561	0.0864	0.1229
Folk Indie R&B	0.0515	0.0227	0.0144	0.1133	0.0237	0.0453	0.5541	0.1751
Jazz	0.0962	0.0253	0.0101	0.0911	0.0142	0.1387	0.2621	0.3623

Figure 5: Genre Classification:Confusion matrix for Linear SVC with TF-IDF

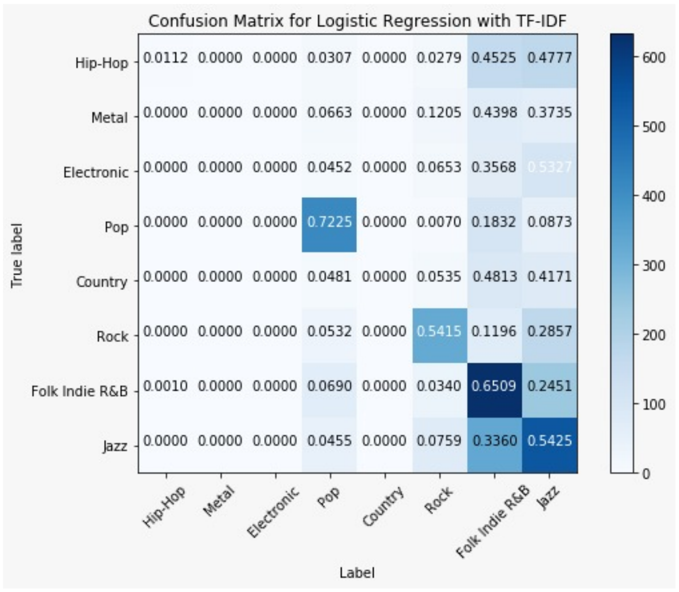


Figure 6: Genre Classification:Confusion matrix for Logistic Regression with TF-IDF

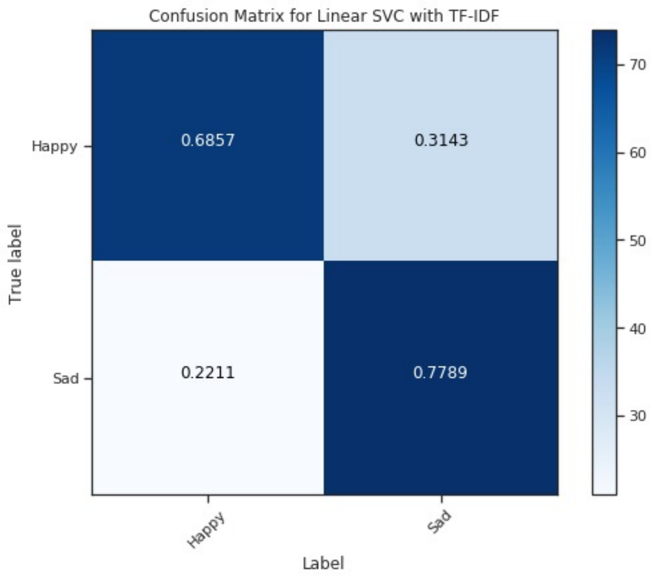


Figure 7: Mood Classification:Confusion matrix for Linear SVC with TF-IDF

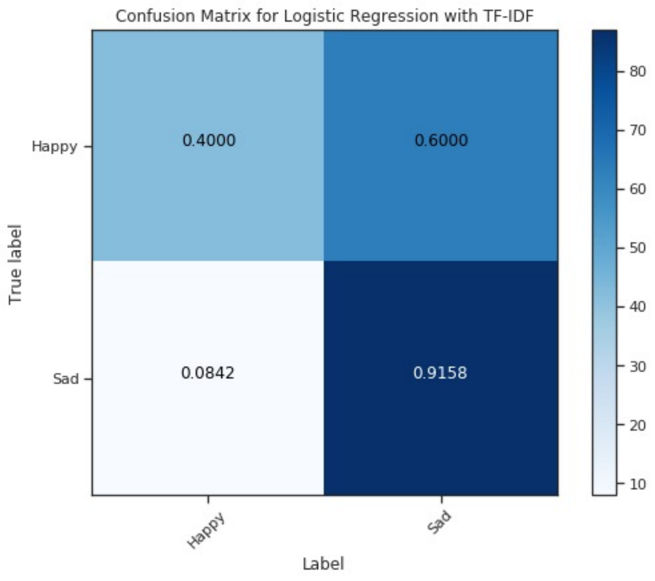


Figure 8: Mood Classification:Confusion matrix for Logistic Regression with TF-IDF

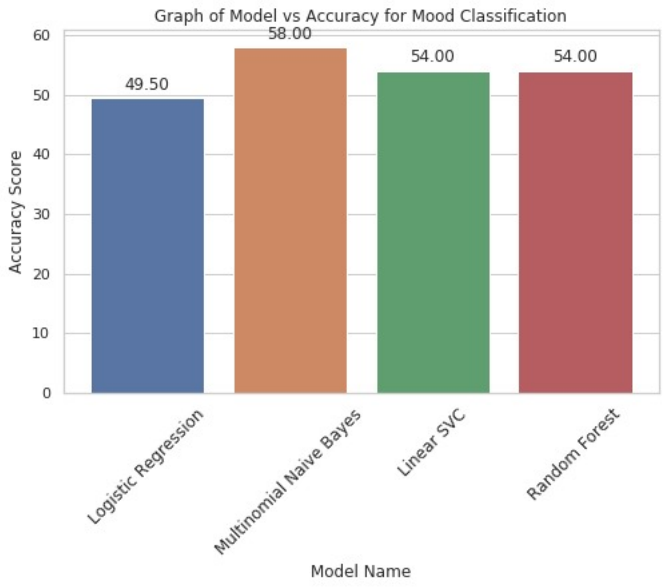


Figure 9: Baseline Model comparison for Mood Classification

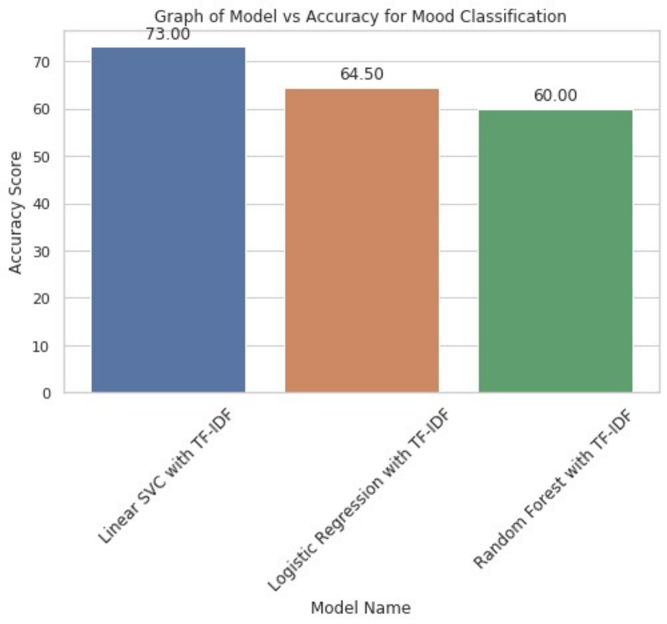


Figure 10: TF-Idf model comparison for Mood Classification

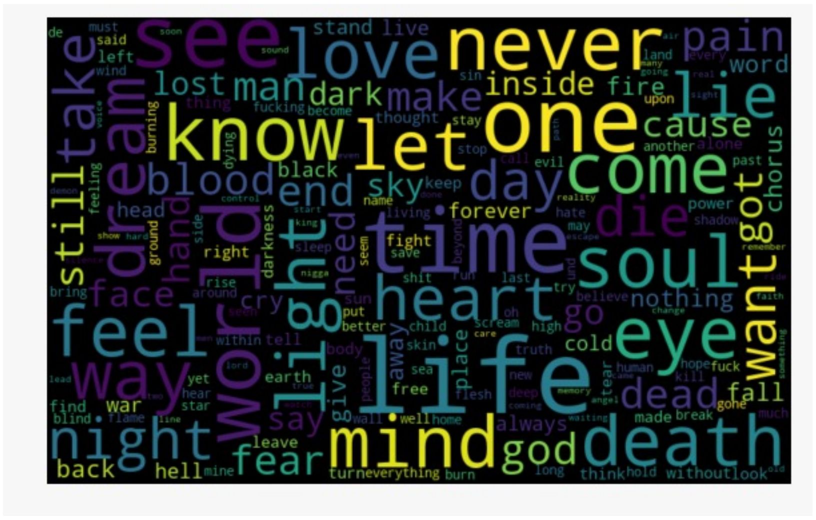


Figure 11: Metal Genre

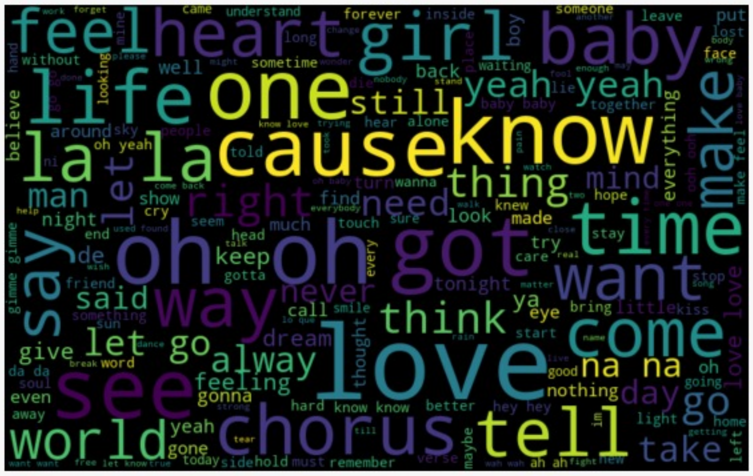


Figure 12: Pop Genre

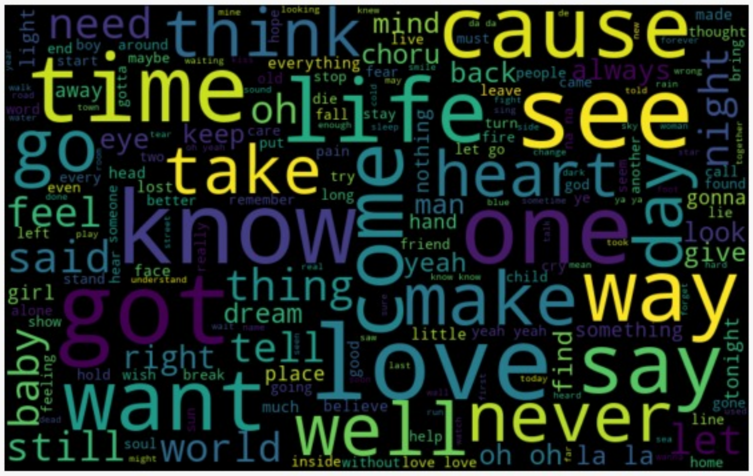


Figure 13: Rock Genre

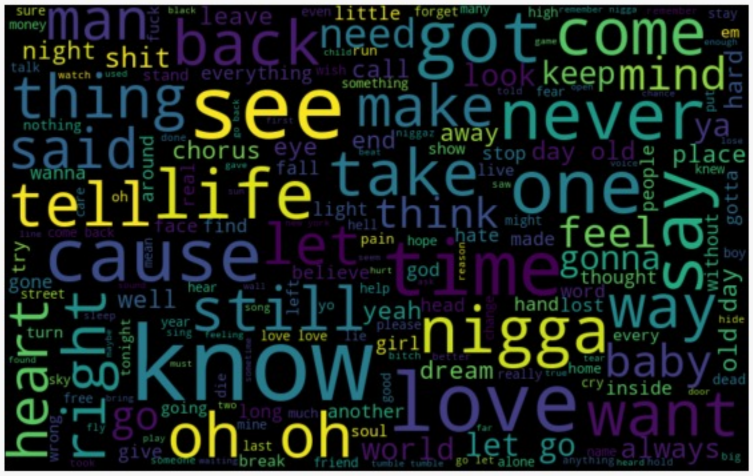


Figure 14: Sad Mood

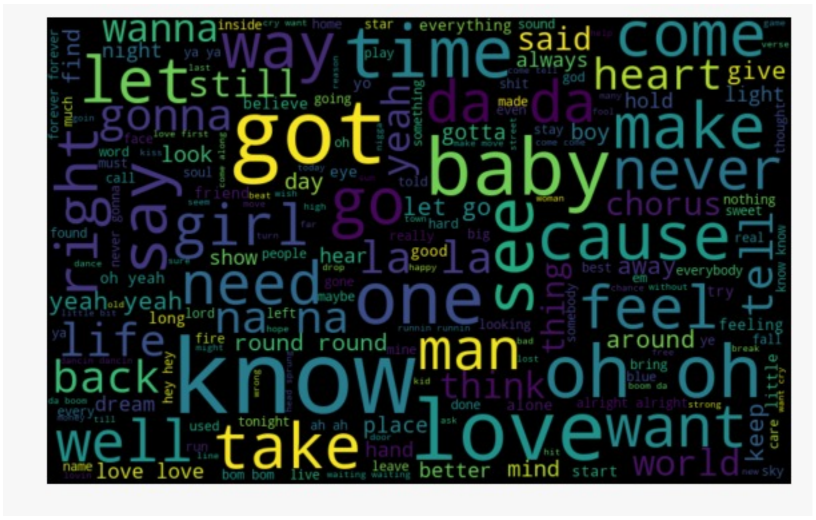
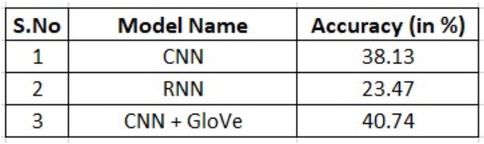
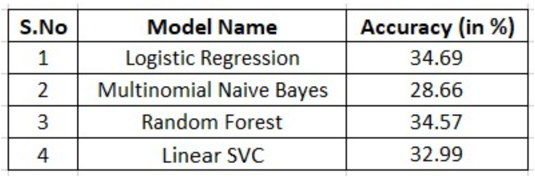


Figure 15: Happy mood



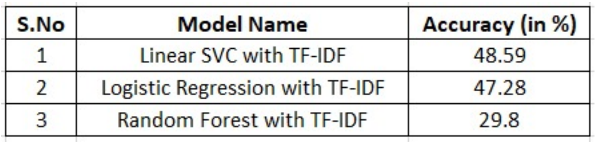
S.No	Model Name	Accuracy (in %)
1	CNN	38.13
2	RNN	23.47
3	CNN + GloVe	40.74

Figure 16: Neural net performance for Genre Classification



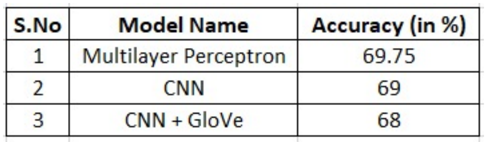
S.No	Model Name	Accuracy (in %)
1	Logistic Regression	34.69
2	Multinomial Naive Bayes	28.66
3	Random Forest	34.57
4	Linear SVC	32.99

Figure 17: Baseline summary for Genre Classification



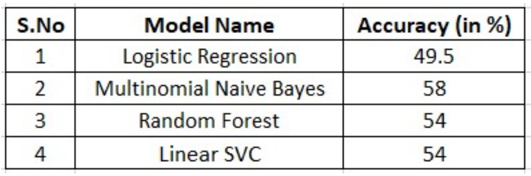
S.No	Model Name	Accuracy (in %)
1	Linear SVC with TF-IDF	48.59
2	Logistic Regression with TF-IDF	47.28
3	Random Forest with TF-IDF	29.8

Figure 18: TF-IDF model comparison for Genre Classification



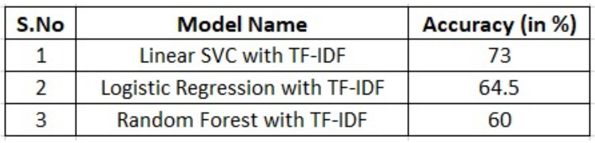
S.No	Model Name	Accuracy (in %)
1	Multilayer Perceptron	69.75
2	CNN	69
3	CNN + GloVe	68

Figure 19: Neural net performance for Mood Classification



S.No	Model Name	Accuracy (in %)
1	Logistic Regression	49.5
2	Multinomial Naive Bayes	58
3	Random Forest	54
4	Linear SVC	54

Figure 20: Baseline summary for Mood Classification



S.No	Model Name	Accuracy (in %)
1	Linear SVC with TF-IDF	73
2	Logistic Regression with TF-IDF	64.5
3	Random Forest with TF-IDF	60

Figure 21: TF-IDF Model comparison for Mood classification

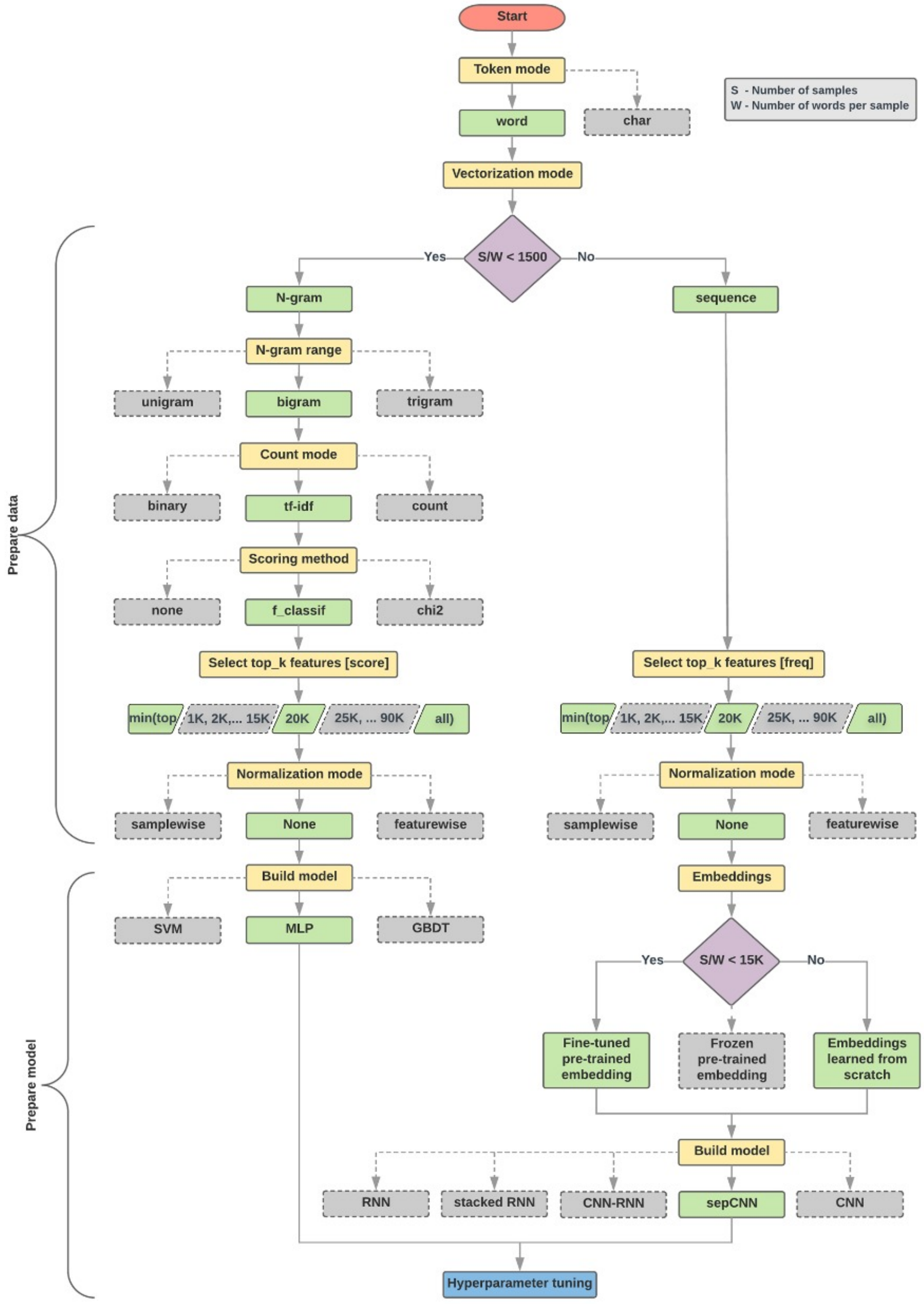


Figure 22: Google Text Classification Flowchart

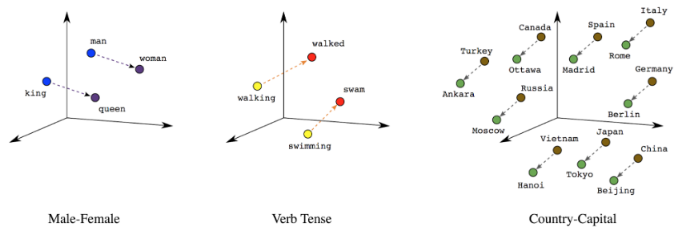


Figure 23: Word Embeddings

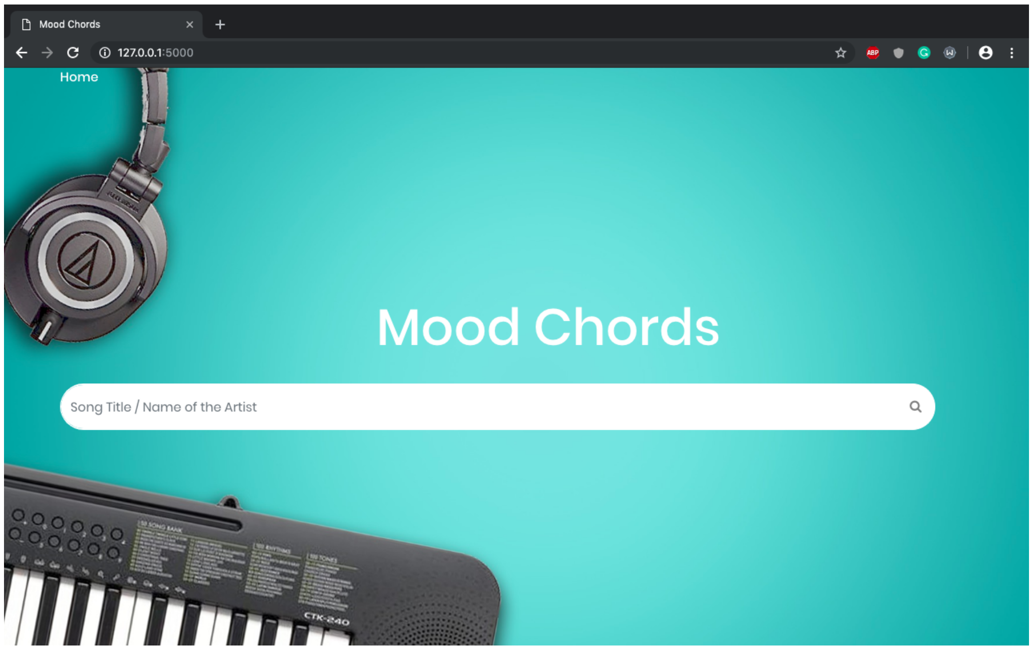


Figure 24: Homepage

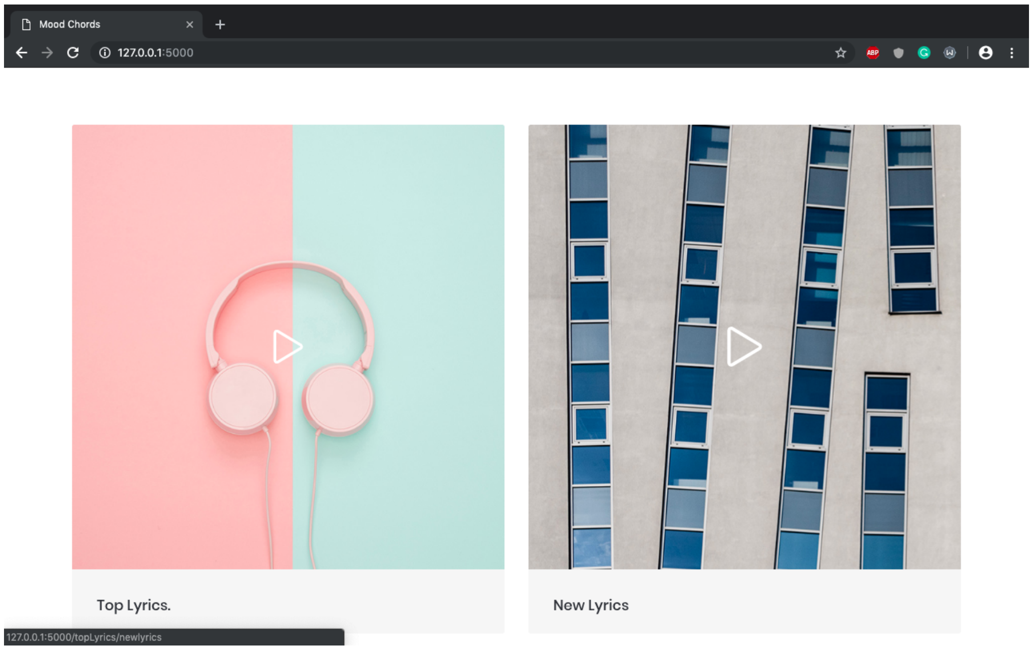
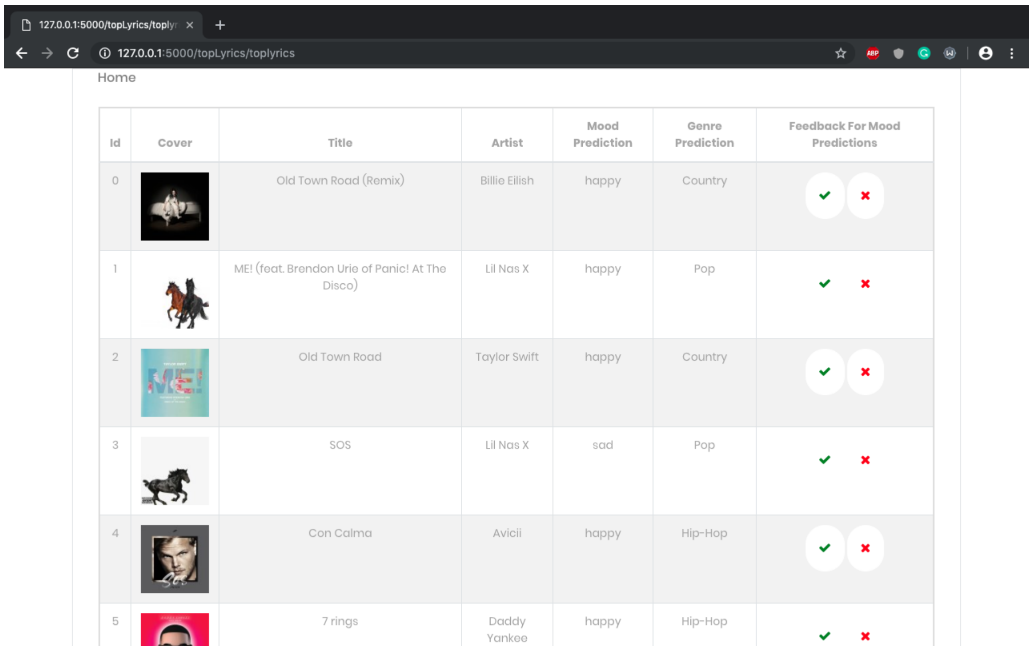


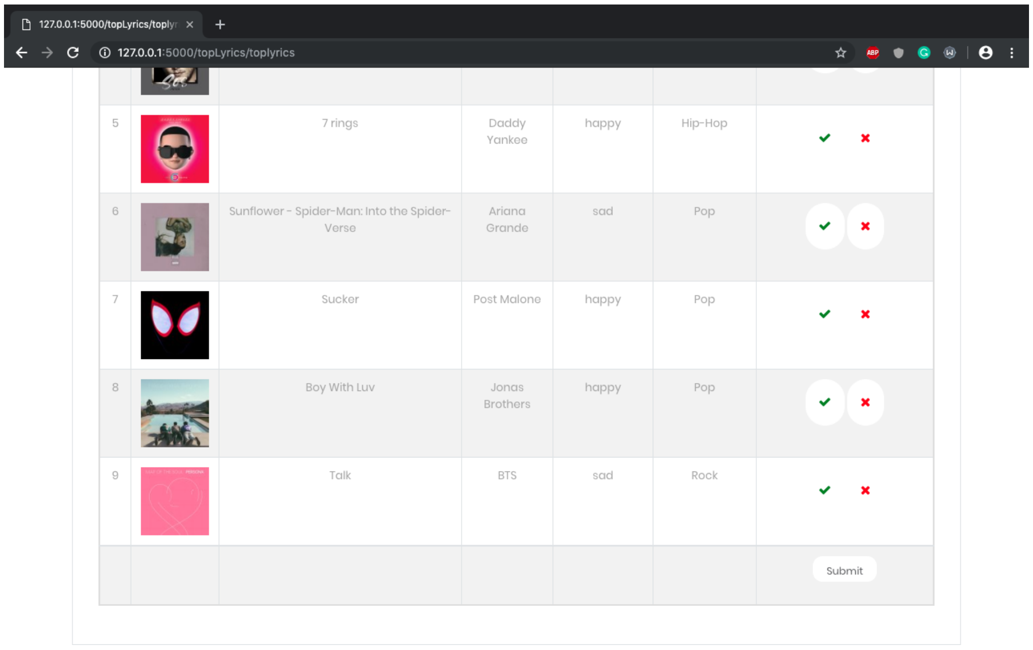
Figure 25: Homepage


127.0.0.1:5000/topLyrics/toplyrics

Home

Id	Cover	Title	Artist	Mood Prediction	Genre Prediction	Feedback For Mood Predictions
0		Old Town Road (Remix)	Billie Eilish	happy	Country	✓ ✗
1		ME! (feat. Brendon Urie of Panic! At The Disco)	Lil Nas X	happy	Pop	✓ ✗
2		Old Town Road	Taylor Swift	happy	Country	✓ ✗
3		SOS	Lil Nas X	sad	Pop	✓ ✗
4		Con Calma	Avicii	happy	Hip-Hop	✓ ✗
5		7 rings	Daddy Yankee	happy	Hip-Hop	✓ ✗

Figure 26: Prediction table


127.0.0.1:5000/topLyrics/toplyrics

5		7 rings	Daddy Yankee	happy	Hip-Hop	✓ ✗
6		Sunflower - Spider-Man: Into the Spider-Verse	Ariana Grande	sad	Pop	✓ ✗
7		Sucker	Post Malone	happy	Pop	✓ ✗
8		Boy With Luv	Jonas Brothers	happy	Pop	✓ ✗
9		Talk	BTS	sad	Rock	✓ ✗
						Submit

Figure 27: Prediction table with Submit

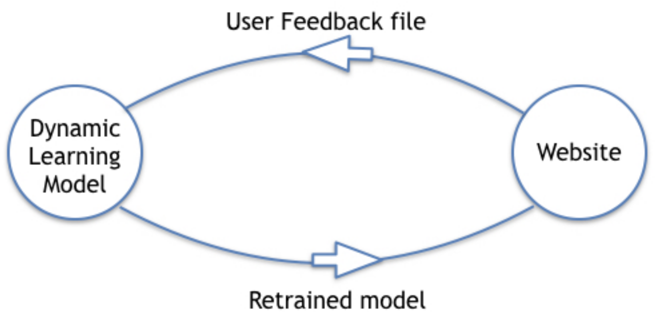


Figure 28: Google Text Classification Flowchart