Halifax, Nova Scotia, Canada

CSCI6505: Machine Learning

Project Final Report

By Team Members:

Naga Pavan Srivathsav Chevuru – B00893685
Nikita Kothari - B00893710
Tanvi Paruthi - B00875949
Tasnim Khan - B00882598

Table of Contents

Project Motivation

During the exploration process of ideas for our course project, we discovered how we can use music data and its features to create moods in tracks and, as a result, relate to the emotions of people. Also, by associating our love for music with the algorithms we learn throughout the course, we would be motivated to work hard and keep looking for more. Those are the thoughts behind this idea. A general music clip or an audio clip can belong to multiple classes or labels such as happy and energetic, sad and lonely, and so on. We will therefore implement a multi-label classification algorithm so that a song can be classified into multiple labels to get better insights. Then, we could use this learning and behavior to determine a population's mood at a given location. We want to cite this article[1] that played a crucial role in motivating us to take this as our course project.

Data Preparation

Below are the two datasets that we have gathered and prepared for the project:

Kaggle Data

Part of the dataset for our project "Music Mood Classification" is taken from Kaggle. It consists of more than 32 thousand songs, their features and classifies them into happy and sad moods.

Spotify Data

Since the data from Kaggle only classified the songs as happy and sad, we extracted more labeled data from Spotify using Spotify public APIs. Since the data from Kaggle only classified the songs as happy and sad, we extracted more labeled data from Spotify using Spotify public APIs. To get more songs labeled with different emotions, we did a manual search. We went to Spotify search, searched for different moods, got the URI from the playlist URL, and used it in our Python script to get the data for those URIs from Spotify. We got 50 playlists URI with 6 different moods, which we used to get data of over three thousand songs. This gave us more mood labels like calm, dark, chill, and so on.
Data from Kaggle and Spotify, combined will be our ground truth.

Data Exploratory Analysis

We performed exploratory analysis on the dataset and plotted different visualizations for understanding the correlation between the datasets and understand the pattern.

The figure 01 shows the features correlation in the dataset.

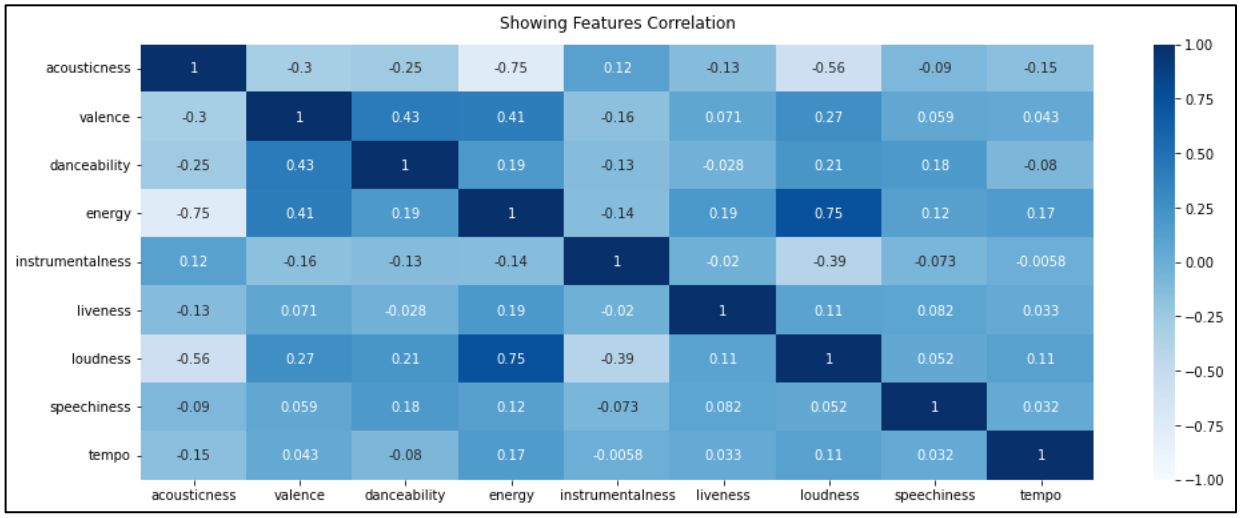

Fig 01: Heatmap showing the feature correlation

With the analysis, the observations that we could make were:

1. We observed that the energy and loudness were highly proportional and correlated.
2. Valence was equally related to danceability and energy.
3. Minor Features Seggregation: Instrumentalness, liveness, Speechiness, and tempo, acousticness
4. Major Features Seggregation: Energy, loudness, valence, danceability

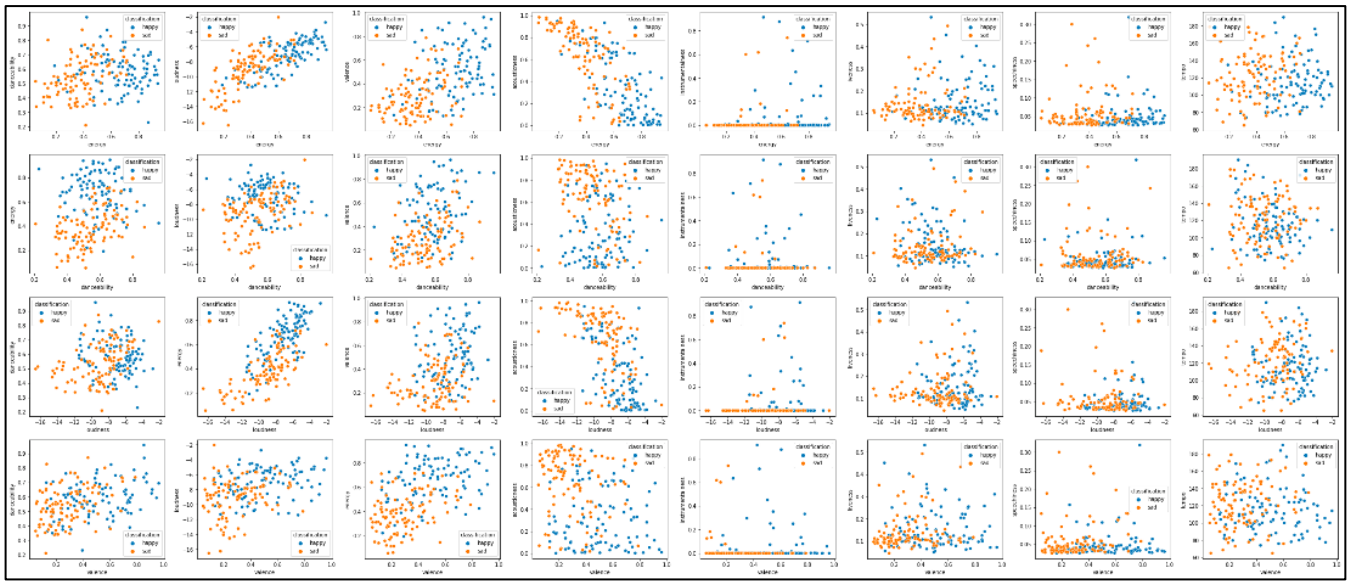

Fig 02: ***Scatter Plot of the Kaggle Data to classify happy and sad songs based on the different features.***

From the above scatter plots, the list of observations that we were able to identify:

1. Energy is low for the Sad songs, and high for the happy songs.
2. Instrumentalness is 0 almost for all songs and hence this feature can be dropped out.
3. Valence is spread out.
4. Danceability for most of the sad and happy songs lie between 0.45 to 0.75
5. Most songs are low on liveness and speechiness

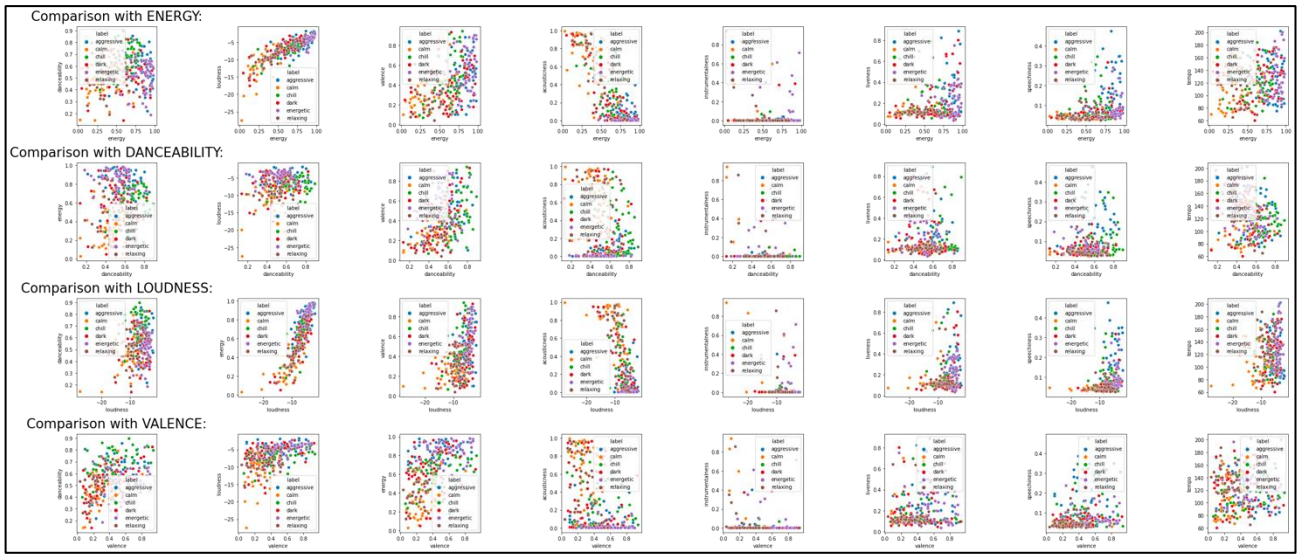

Fig 03: ***Scatter Plot of the Spotify Data to classify different emotions based on the unique features.***

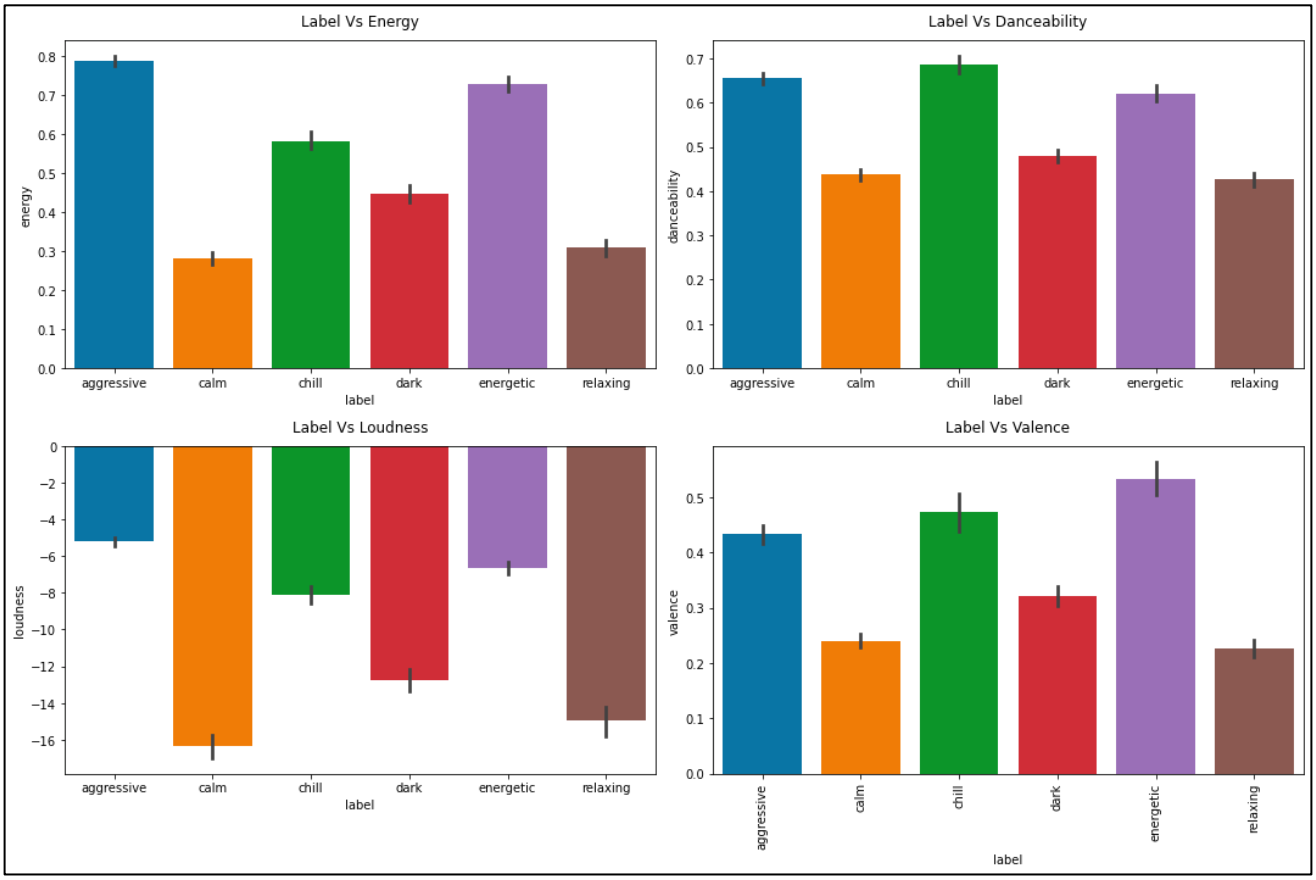

Fig 04: ***Bar plot for highly correlated features***

From the above scatter plots, we were able to get similar results as the one we got from happy and sad songs labeled data from Kaggle.

SMOTE for Classification

Since we had unbalanced datasets, the model will have hard time learning the decision boundaries between the classes. Hence, we used SMOTE to balance the records across the labels. We used imbalanced-learn Python library that provides different implementations of approaches to deal with imbalanced datasets.

Smote would perform below steps:
1) Selecting a random example from the minority class
2) Finding the k (typically k = 5) nearest neighbors of that example
3) Selecting a random example from those neighbors
4) Drawing a line between those two examples
5) Generating a synthetic example by choosing a random point from that line

Drawback of SMOTE: It does not have any knowledge regarding the underlying distribution. Therefore, some noisy samples can be generated. As a result, we concluded that we would be using another sampler that imbalanced-learn provides. Out of the two samplers, SMOTETomek and SMOTEENN, we chose to proceed by using SMOTETomek.

Note: If we re-run the model again to compare the results, there might be some change in numbers due to random initialization.

Without Dataset Resampling		With Dataset Resampling	
Model Name	Accuracy	Model Name	Accuracy
Logistic Regression	35.57	Logistic Regression	30.31
Naïve Bayes	35.31	Naïve Bayes	40.16
KNN	30.48	KNN	60.75
Decision Tree Classifier	32.46	Decision Tree Classifier	50.05
Random Forest Classifier	45.68	Random Forest Classifier	66.26
SVC	29.7	SVC	23.37

Fig 05: ***The comparison of model results before and after resampling using SMOTE***

Synthetic Data Generation

We had unbalanced dataset where we had emotion labels of happy and sad songs to be over 32 thousand and the other labels combined to be a little over 3 thousand. We decided to generate our synthetic data using Generative Adversarial Network (GAN) Model.
GAN model is an unsupervised learning algorithm where there are two neural networks, the generator generates synthetic data, and the discriminator tries to classify the given data as real or synthetic. If the discriminator correctly identifies the synthetic data, the generator is penalized, and if discriminator does not identify the data, then it is penalized. Thus, both the neural networks compete to provide more accurate synthetic data.
This model was not used in our project as we were not able to improve the accuracy of the synthetic data and the analysis did not produce similar results as the real data.

Classic ML Model

Overfitting of the Model

When we train a model with a subset of training data, the model then fails to learn and generalize. Then it causes the model to achieve an extremely high training accuracy while training but when we use the model to predict values from unseen data then it fails as the model has not generalized enough. The over-fitting can also be done when you use hyper parameters which are way above what is required. For example: Number of clusters in KNN, Max Depth in Decision Tree etc. This is what we have done to all the classical machine learning models below. We have taken a subset of data and tuned hyper parameters to overfit it.

KNN Model

The KNN algorithm is a supervised learning algorithm used for both regression and classification. It assumes that similar datapoints exist near each other. It tries to predict the correct class for the data by calculating the distance between the test data and all the training points. It then selects the K number of points that is closest to the data and then classifies based on probability calculation.

The KNN algorithm predicts the values of new datapoints based on 'feature similarity,' which implies that the new data point will be assigned a value depending on how closely it resembles the points in the training set.

Basic steps of KNN:

Step 1: We need a dataset to implement any algorithm. As a result, we must load both training and test data at the first stage of KNN.
Step 2: The value of K, i.e., the closest data points, must then be chosen. Any integer can be used as K.
Step 3: Do the following for each point in the test data:

- **3.1:** Calculate the distance between each row of training data and the test data using one of the following methods: Euclidean, Manhattan, or Hamming distance. The Euclidean technique is the most widely used method for calculating distance.
- **3.2:** Sort them in ascending order depending on the distance value.
- **3.3:** The top K rows of the sorted array will then be chosen.
- **3.4:** The test point will now be assigned a class based on the most common class of these rows.

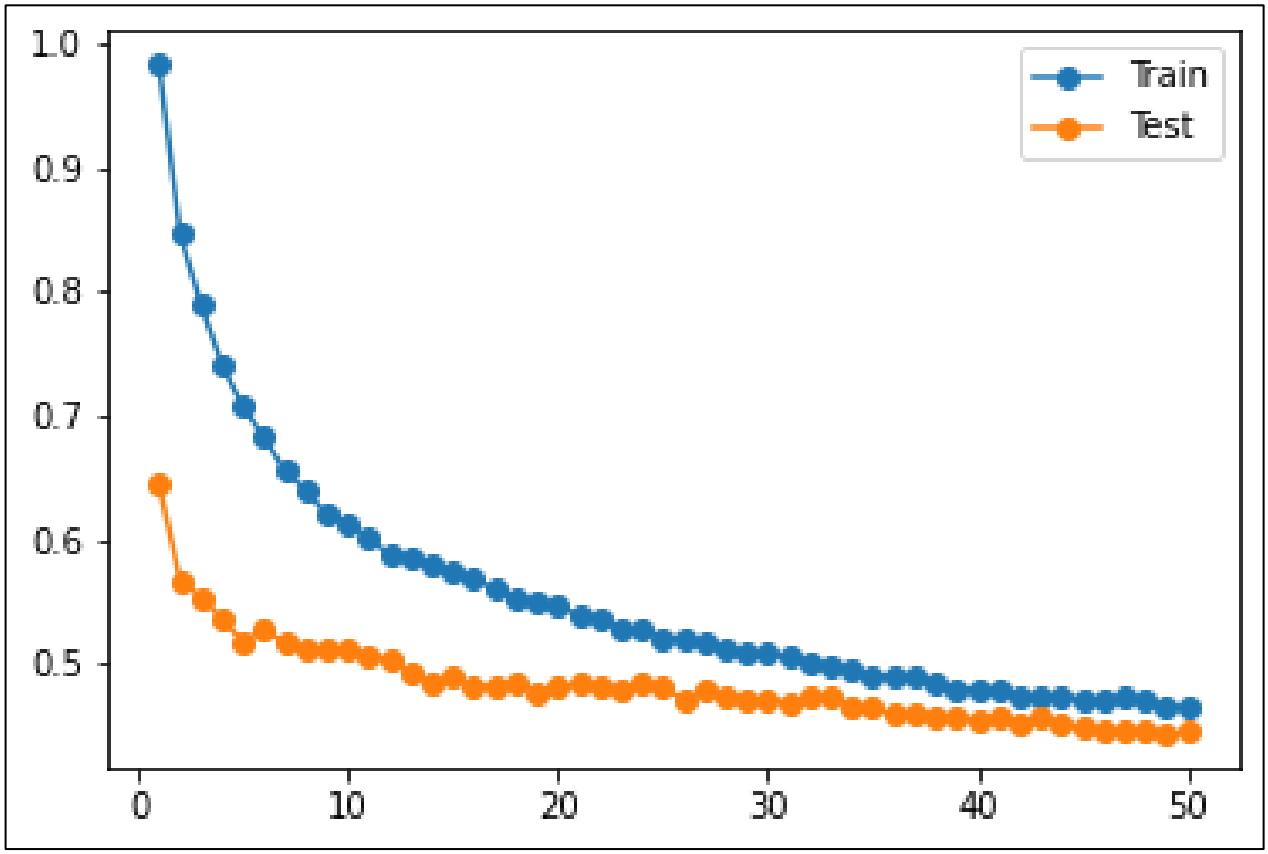

Fig 06: Accuracy comparison of our data using KNN Model for Overfitting

As expected in overfitting, we can see that the training accuracy is higher than the testing accuracy.

Decision Tree Model

The most powerful and widely used tool for classification and prediction is the decision tree. A decision tree is a flowchart-like tree structure in which each internal node represents an attribute test, each branch reflects the test's conclusion, and each leaf node stores a class label.

Construction

By separating the source set into subgroups based on an attribute value test, a tree can be "trained." Recursive partitioning is the process of repeating this method on each derived subset. When all the subsets at a node have the same value of the target variable, or when splitting no longer adds value to the predictions, the recursion is complete. Because the building of a decision tree classifier does not need domain expertise or parameter selection, it is well suited to exploratory knowledge discovery. High-dimensional data can be handled via decision trees. The accuracy of the decision tree classifier is good. A common inductive strategy to learning classification information is decision tree induction.

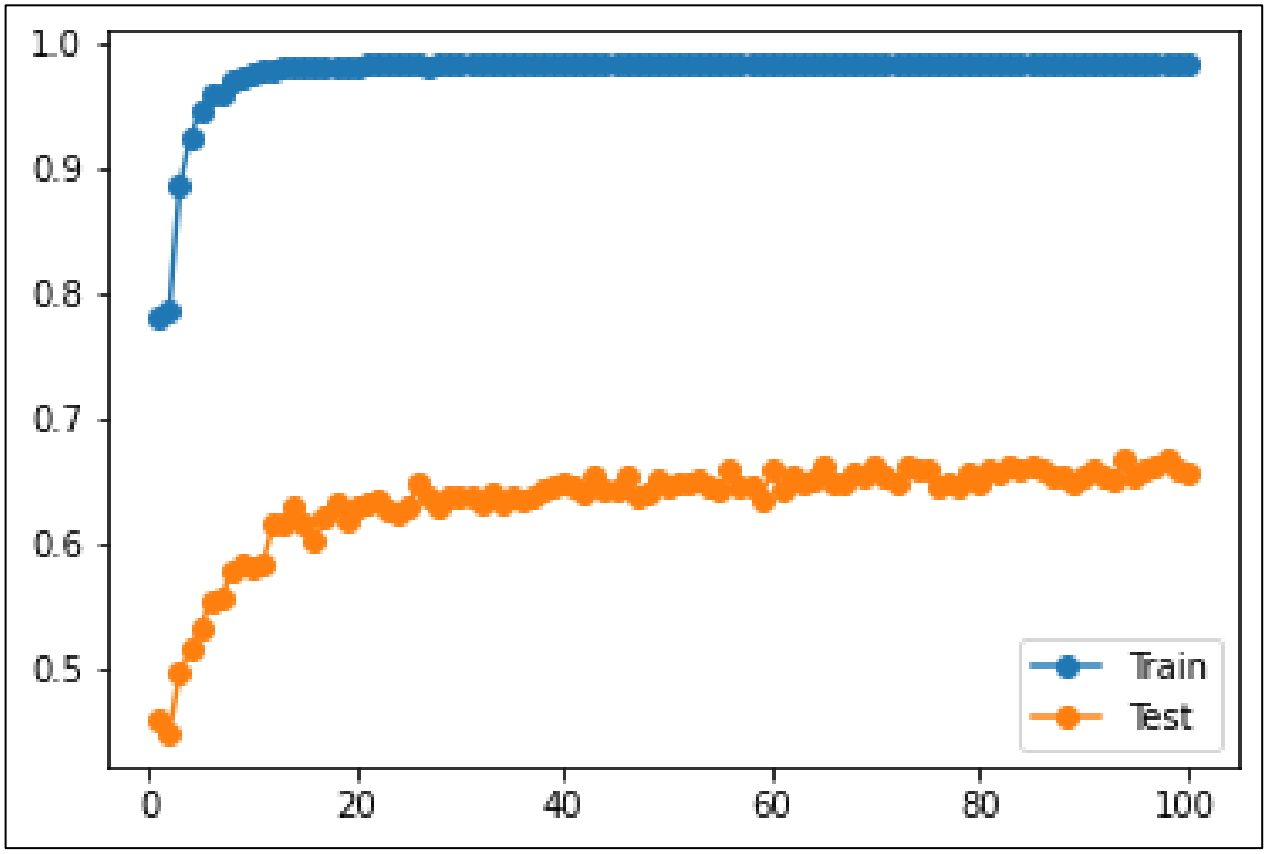

Fig 07: ***Accuracy comparison of our data using Decision Tree Model for Overfitting***

We can see from the above graph that there is a huge gap between the training and testing accuracy curves. This huge area between the graphs shows that the model is overfitting the data.

RFC Model

The Random Forest Classifier is a combination of several decision trees. It is a classification algorithm which uses bagging and features randomness when building a particular decision tree and then creates several of these decision trees to create a forest of trees. This collection of decision trees has more accuracy than the individual decision tree.

- **Using NEstimator Hyperparameter:**

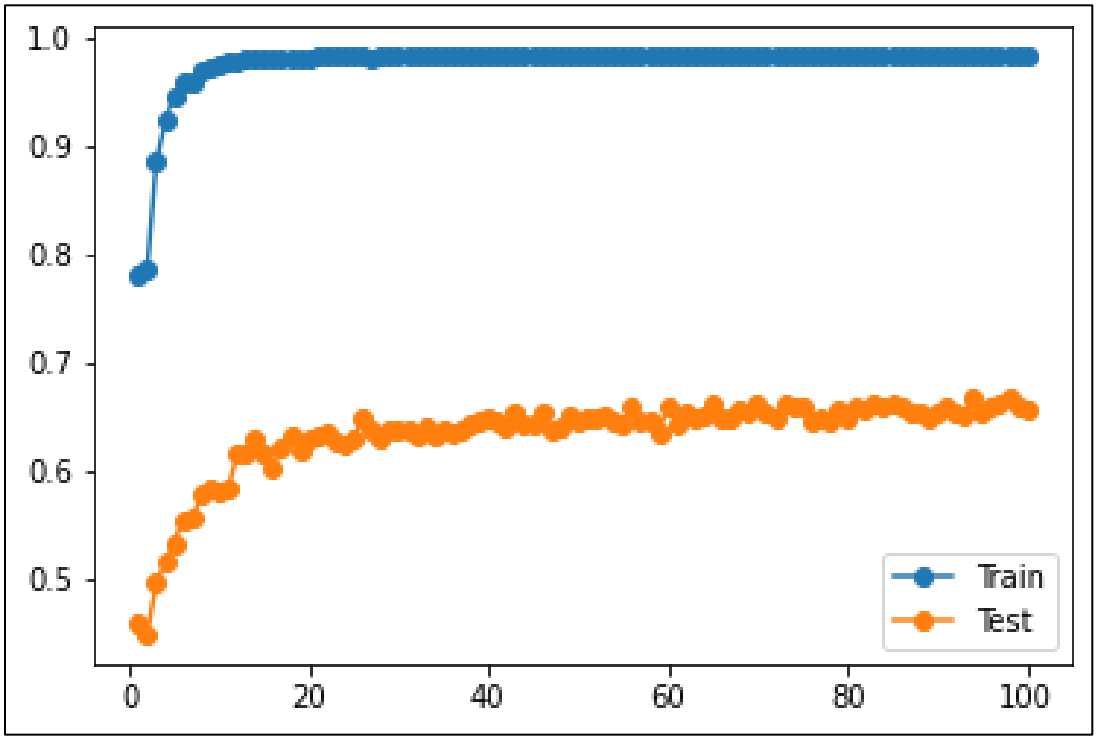

Fig 08: Accuracy comparison of our data using RFC Model (NEstimator Hyperparameter) for Overfitting

- **Using MAX Depth hyperparameter:**

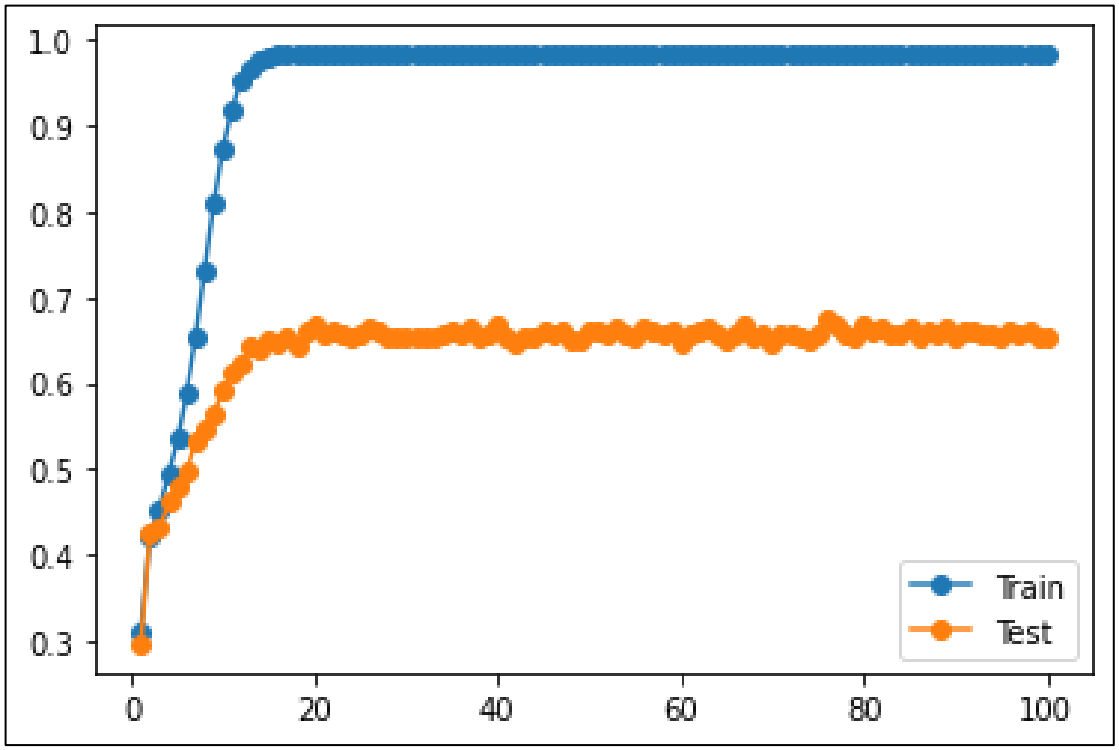

Fig 09: ***Accuracy comparison of our data using RFC Model (Max Depth Hyperparameter) for Overfitting***

Tuning Classical ML Models

Once we had the basic models then we tried improving those models by identifying the right combinations of the hyper parameters which would give the model the highest accuracy that can be achieved on the dataset. For this we have used the GridSearchCV from the sklearn library. This library looks at all the permutations and combinations of various hyper meters to get the right combination for each algorithm. Then we stored the best parameters and passed them to train the model again which improved the performance of the models.

We also performed PCA to check the model performance after doing the PCA analysis. Even though our model didn't have many features to perform the dimensionality reduction, we thought that this might combine the highly correlated features which might help us increase the performance of the model. All the results of these models have been shown in the respective .ipynb files along with any graphs.

ANN Baseline Model

Artificial Neural Networks are made up of artificial neurons known as units. The Artificial Neural Networks of a system are made up of these units, which are stacked in a succession of layers. The number of units in a layer can range from a few hundred to millions, depending on the system's complexity. An Artificial Neural Network typically contains an input layer, an output layer, and hidden layers. The input layer gets information from the outside world that the neural network must interpret or learn. The data then flows via one or more hidden layers, which converts the input into useful data for the output layer. Finally, the output layer generates a response from the Artificial Neural Network.

Baseline Model

In our baseline model we have used a very basic architecture where our model has one hidden layer having ReLU activation and the output layer with the SoftMax activation to calculate the logarithmic loss based on the cross-entropy loss. Furthermore, initially the model is trained with the Adam optimizer with a 0.001 learning rate.

Data Sampling

As explained earlier, we have an unbalanced dataset. The model will have a hard time learning the decision boundaries between the classes. Hence, we used SMOTE-Tomek to balance the records across the labels

Data Scaling + Data Sampling

Our dataset had nine features that were measured at different scales. Now, variables with different scales do not contribute equally to the model fitting and might end up creating a bias. Thus, to deal with this potential problem we decided to implement data scaling. We used Standard scalar on top of the data sampling technique used. StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way.

Three Hidden layers added along with Data Sampling and Scaling

To make the data non-linearly separable, we added three hidden layers in our neural network architecture. We used RelU activation function along with these hidden layers and validated the performance of our model. The below comparative analysis shows the model accuracy as we build up the architectures.

Comparative Analysis

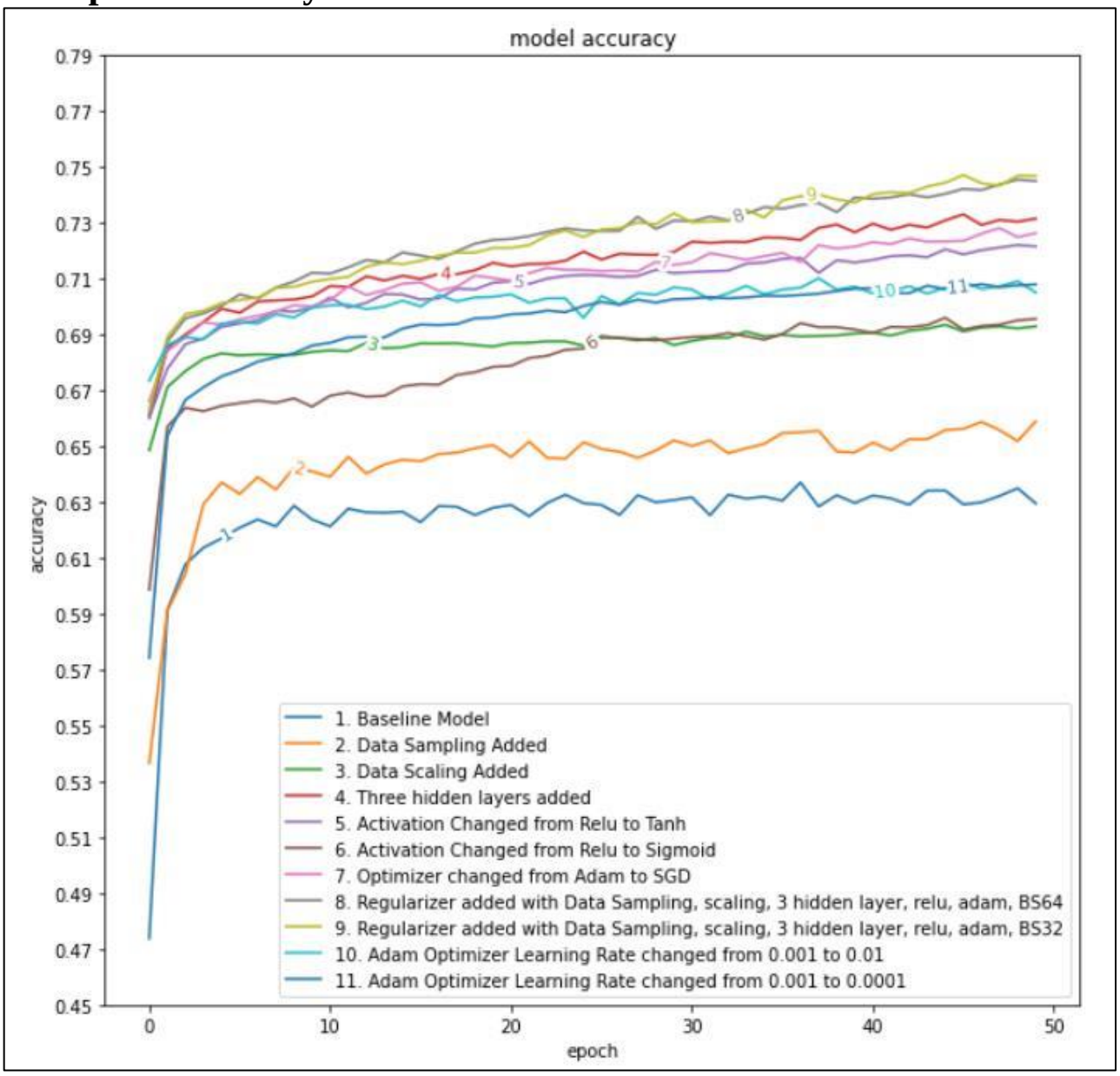

Fig 10: ***Model Accuracy comparison chart***

Activation Changed from Relu to Tanh (Three Hidden layers, Data Sampling and Scaling)
ReLU solves the problem of vanishing gradients and allowed neural networks to be of large sizes, hence it has accuracy better than Tanh.

Activation Changed from Relu to Sigmoid (Three Hidden layers, Data Sampling and Scaling)
ReLU solves the problem of vanishing gradients and allows neural networks to be of large sizes, while, in sigmoid, during backpropagation, the gradients diminish in value when they reach the beginning layers. This stopped the neural network from scaling to bigger sizes with more layers. Therefore, the accuracy of ReLU is better than Sigmoid.

Optimizer Changed from ADAM to SGD (Three Hidden layers, Data Sampling and Scaling)

SGD only computes on a limited subset or random selection of data instances, rather than doing computations on the entire dataset, which is redundant and inefficient. When the learning rate is modest, SGD delivers the same results as normal gradient descent.

Regularizer Added (Three Hidden layers, Data Sampling and Scaling, Adam Optimizer)

Adding regularizes shrinks the coefficient estimates towards zero, thereby discouraging the learning of a more complex model and avoiding the risk of overfitting. Since we have some features that are correlated with each other, regularizer avoids overfitting and thus the accuracy is increased

Batch Size changed to 32 (Three Hidden layers, Data Sampling and Scaling, Adam and regularizer)

While the model tends to provide faster convergence to a good solution, having smaller batch size does not guarantee the converge of the model to global optima. It may bounce around the global optimum to a local optimum.

Learning Rate

The learning rate is a parameter that determines how quickly the model adapts to the situation. Smaller learning rates need more training epochs given the more minor changes to the weights each update. In contrast, bigger learning rates necessitate fewer training epochs. A large learning rate can cause the model to converge too rapidly to a suboptimal solution, whereas a too-low learning rate can cause the process to stall.

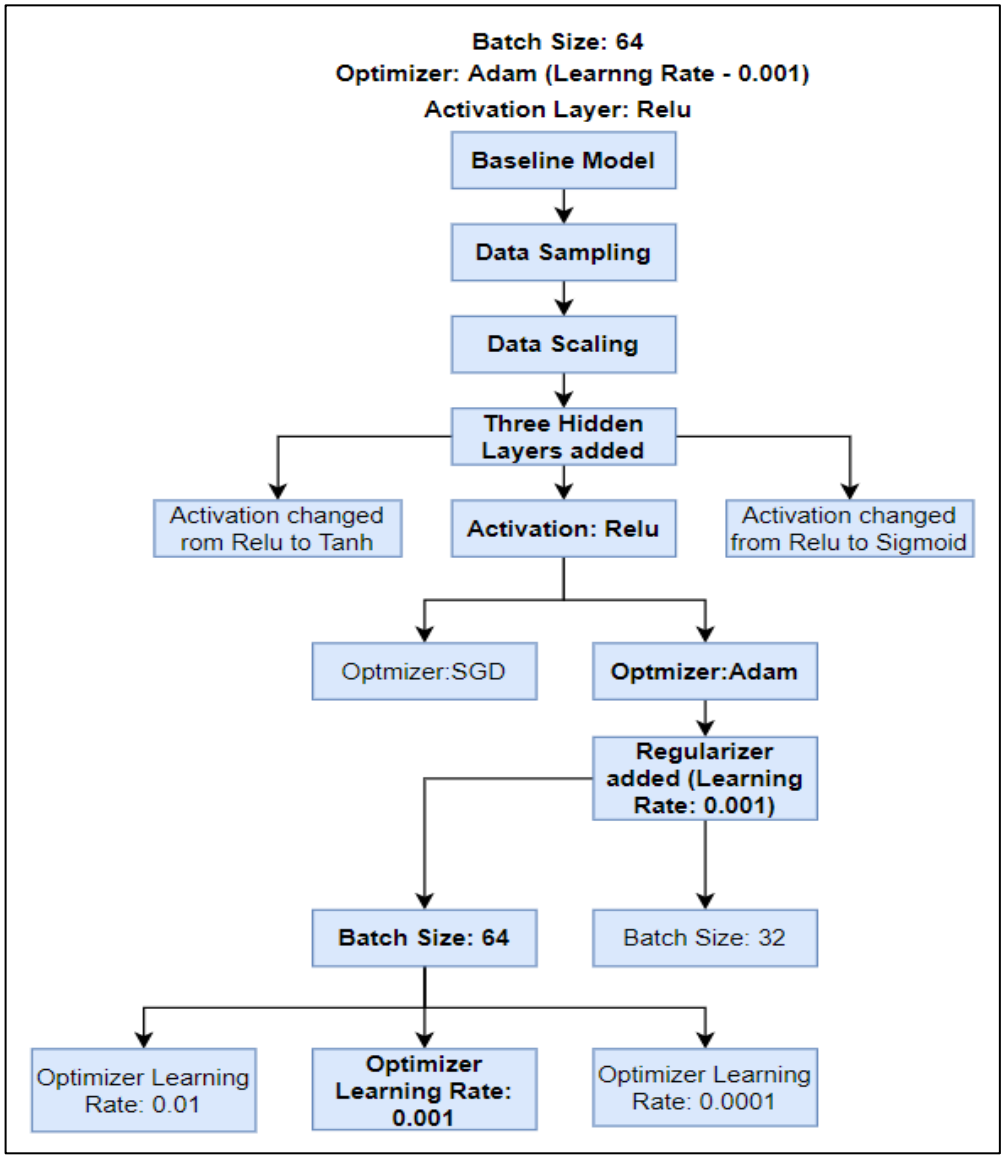

Fig 11: ***Production ANN Model Architecture***

Analysis of Misclassified Songs Data

We randomly chose 10 happy songs and 10 sad songs that were misclassified.
The observations or the patterns that we could find in the songs that were wrongly classified as happy songs were:

1. Acousticness, dancibiliity or the energy were high for these songs.
2. When listening to these actual songs, we found out that the lyrics were sad, but had good beats or high energy. For example, the sad song was actually a Jazz or Metal song, which have high energy and acousticness, respectively.
3. The song classified by Spotify is based on lyrics for these, whereas the model classifies the songs, based on the music data that we extracted.

The observations or the patterns that we could find in the songs that were wrongly classified as sad songs were:

1. Acousticness, dancibiliity, energy or liveness were low for these songs.
2. When listening to these actual songs, we found out that the lyrics were happy or motivational, but the music was soothing, or concentration music, which lacked energy and dancibility.
3. The song classified by Spotify is based on lyrics for these or the motivation, whereas the model classifies the songs, based on the music data that we extracted.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Uri	Song Url	acousticness	valence	danceability	energy	instrumentalness	liveness	loudness	speechiness	tempo	h/s	Prediction	Mistakes	Explanation	Actual Song	Y_label
2	spotify:track:5oM9F	https://open.spotify.cor	0.624	0.661	0.623	0.438	0	0.0886	-9.336	0.0475	114.283	s	h	Incorrect	Acousticness, Dancibility	Sad Lyrics, Energetic Music, Jazz	1
3	spotify:track:0AuQz	https://open.spotify.cor	0.85	0.146	0.7	0.304	0.0258	0.063	-13.191	0.18	80.027	s	h	Incorrect	Acousticness, Dancibility	Sad Lyrics, Sad Music, Great Beats	1
4	spotify:track:5TxPQ	https://open.spotify.cor	9.05E-05	0.0384	0.272	0.962	0.52	0.0717	-5.211	0.234	118.453	s	h	Incorrect	Energy	Sad Lyric, Metal - High Energy	1
5	spotify:track:4FEYxF	https://open.spotify.cor	0.394	0.283	0.694	0.478	0.00598	0.451	-9.879	0.0321	119.865	s	h	Incorrect	Dancibility	Sad Lyrics, Good Beats	1
6	spotify:track:5hMSs	https://open.spotify.cor	0.0406	0.873	0.621	0.873	9.10E-06	0.0824	-3.067	0.0369	97.96	s	h	Incorrect	Dancibility, Energy	Sad Lyrics, Happy Misuc	1
7	spotify:track:0sc9Hi	https://open.spotify.cor	5.44E-05	0.176	0.224	0.98	0.00237	0.335	-4.193	0.155	155.022	s	h	Incorrect	Energy	Metal - High Energy	1
8	spotify:track:3HWzc	https://open.spotify.cor	0.11	0.412	0.715	0.624	0	0.123	-3.046	0.114	158.087	s	h	Incorrect	Dancibility, Energy	Sad Lyrics, Great Beats	1
9	spotify:track:5RsH8I	https://open.spotify.cor	0.489	0.361	0.655	0.426	0.000779	0.151	-10.86	0.0563	156.949	s	h	Incorrect	Dancibility	No Lyrics, Okay okay msuic - Don't know why it is classified as sad	1
10	spotify:track:1wMvl	https://open.spotify.cor	0.753	0.501	0.782	0.343	0.00577	0.0844	-16.165	0.0601	99.97	s	h	Incorrect	Acousticness, Dancibility	Non English Lyrics, but the energy seemed like happy	1
11	spotify:track:58YCE	https://open.spotify.cor	6.85E-05	0.142	0.407	0.974	0.017	0.363	-4.302	0.167	160.023	s	h	Incorrect	Energy	Metal with Sad Lyrics	1
12	spotify:track:0KIPa0	https://open.spotify.cor	0.0301	0.449	0.697	0.514	0.535	0.212	-8.626	0.0286	136.071	h	s	Incorrect	Acousticness, Liveliness	Felt like slow music	0
13	spotify:track:4VWzk	https://open.spotify.cor	0.982	0.175	0.38	0.0466	0.943	0.111	-22.267	0.0441	140.729	h	s	Incorrect	Energy, Liveliness	Slow Music	0
14	spotify:track:1GBUK	https://open.spotify.cor	0.00158	0.204	0.554	0.73	1.85E-05	0.117	-6.236	0.0418	118.24	h	s	Incorrect	Acousticness, Liveliness	Motivational song, soothing music	0
15	spotify:track:7oJmlI	https://open.spotify.cor	0.105	0.204	0.445	0.563	0.000271	0.179	-8.904	0.0315	94.978	h	s	Incorrect	Acousticness, Liveliness	Happy Lyrics, soothing music	0
16	spotify:track:6tqIesA	https://open.spotify.cor	0.0685	0.562	0.809	0.613	0	0.107	-8.402	0.175	104.991	h	s	Incorrect	Acousticness, Liveliness	Happy Lyrics, Good Beats, slightly soothing	0
17	spotify:track:1qzF5E	https://open.spotify.cor	0.993	0.0458	0.372	0.00682	0.921	0.108	-29.489	0.04	79.15	h	s	Incorrect	Energy, Liveness, dancibility	Concentration Music	0
18	spotify:track:7DFNE	https://open.spotify.cor	0.385	0.244	0.509	0.538	0	0.104	-7.335	0.0572	75.089	h	s	Incorrect	Acousticness, Liveness	Sad Lyrics, "Let her Go", Definitely a sad song	0
19	spotify:track:4zQFPI	https://open.spotify.cor	0.801	0.197	0.279	0.422	0.00117	0.113	-9.72	0.0297	103.921	h	s	Incorrect	Dancibility, Liveness	Slow Music	0
20	spotify:track:33aKD	https://open.spotify.cor	0.176	0.749	0.723	0.69	0	0.116	-5.29	0.41	109.292	h	s	Incorrect	Acousticness, Liveness	Felt like a Happy, good beat song	0
21	spotify:track:05IBuZ	https://open.spotify.cor	0.942	0.157	0.379	0.306	0.323	0.123	-14.446	0.0305	147.902	h	s	Incorrect	Dancibility, Energy, Liveness	Soothing music	0

Fig 11: ***Screenshot of the misclassified song analysis***

I observed that the song labels by Spotify also considered the lyrics, occasion, and other features which we have not taken into consideration in our project. Spotify playlist classifies the songs whereas our model classifies the music.

Further Model Analysis

Figure 12 and 13 shows the behaviour of the ANN model with dataset having 8 labels and 2 labels respectively.

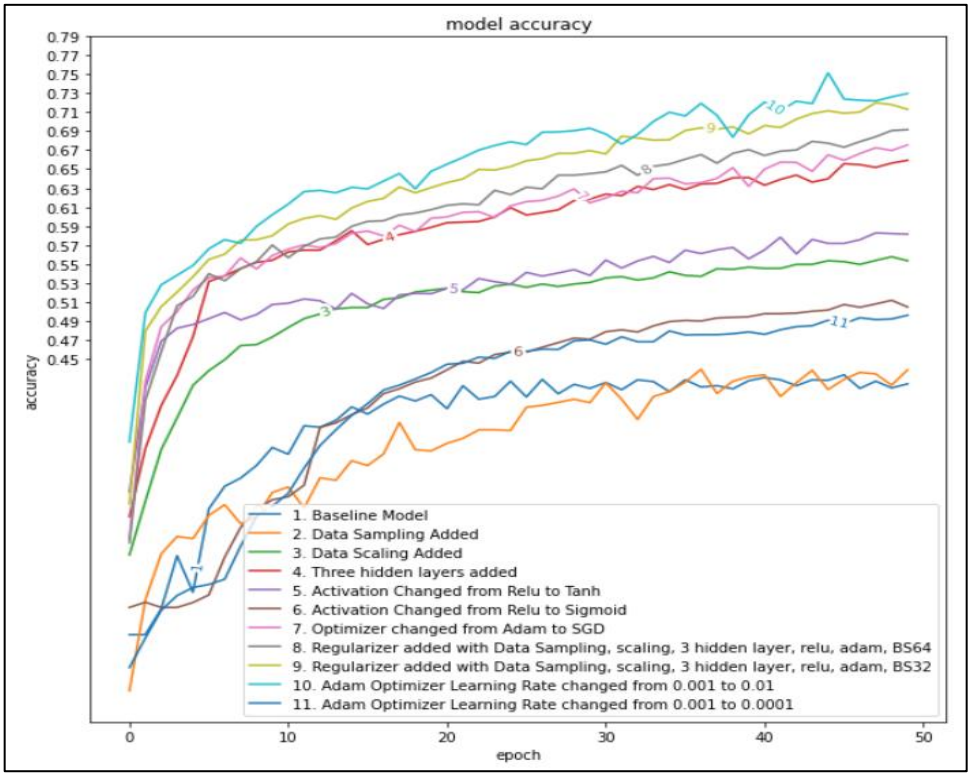

Fig 12: Screenshot of the misclassified song analysis

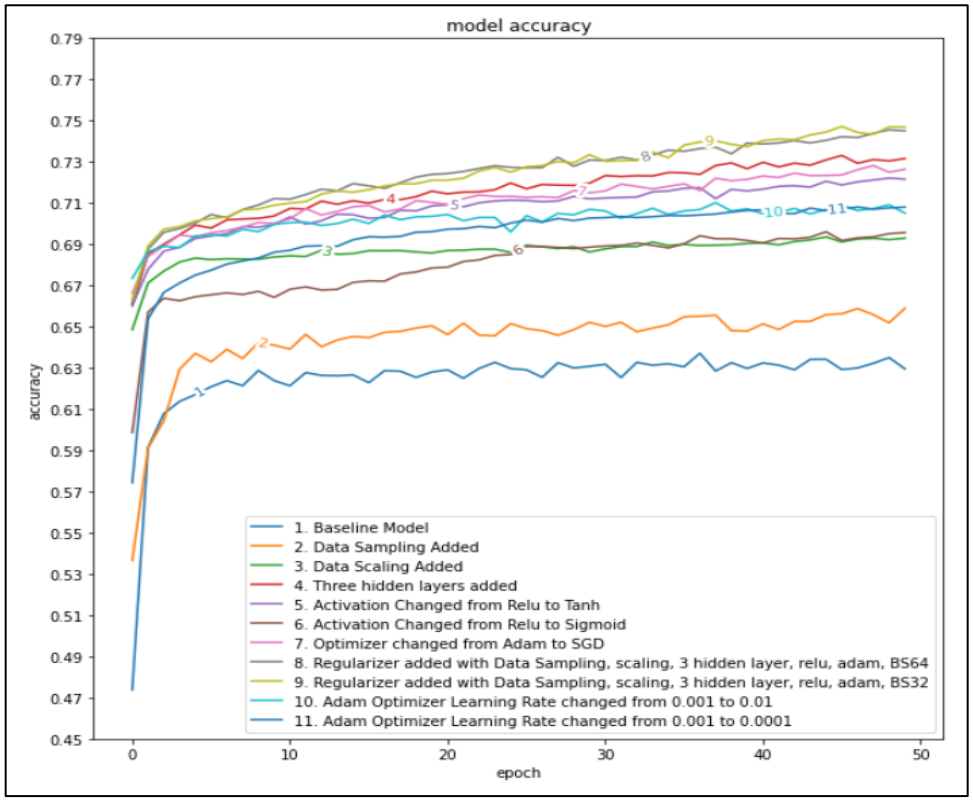

Fig 13: Screenshot of the misclassified song analysis

1. With Data Sampling, the accuracy of the model with dataset having 2 label, happy and sad, increased in comparison with the baseline model.
2. With the increase in learning rate, the dataset with 8 labels performed better as compared to the dataset with 2 labels.
3. The transition from Adam optimizer to SGD optimizer in 8 labels dataset resulted in a better accuracy result.

APPENDIX I

Audio feature explanation

The audio features definitions have been taken from Spotify developer references [2].

1. **Acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
2. **Danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
3. **Energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
4. **Instrumentalness**: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
5. **Liveness**: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides a strong likelihood that the track is live.
6. **Loudness**: the overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
7. **Speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g., talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
8. **Valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g., happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g., sad, depressed, angry).
9. **Tempo**: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, the tempo is the speed or pace of a given piece and derives directly from the average beat duration.

References

[1] A. Edmans, A. Fernandez-Perez, A. Garel, and I. Indriawan, “Music sentiment and stock returns around the world,” SSRN Electronic Journal, 2021.

[2] Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features> [Accessed 12 April 2022].