MUSIC MOODS CLASSIFICATION

CSCI6505: MACHINE LEARNING FINAL PROJECT

DALHOUSIE UNIVERSITY

Motivation

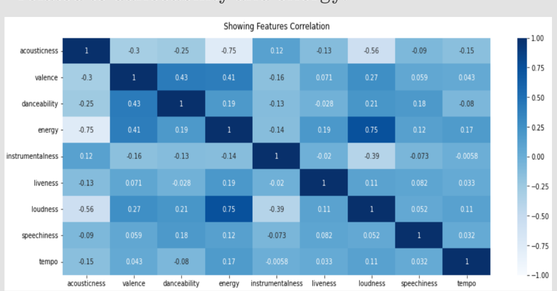
While exploring ideas on which we can work for our course project, we came across how we can leverage the music data and its features to generate the moods of the tracks and, as a result, relate to the emotions of the people. Also, our love for music would keep us motivated to work and look for more by associating it with the algorithms that we learn throughout the course. That is how we thought of working on this idea. Hence, we plan to implement an Artificial Neural Network (ANN) so that we can classify a song into labels to get better insights. Once we achieve a considerable accuracy, we could use this learning and behaviour to determine the population mood at given geolocation. This project essentially focuses on performing a supervised classification algorithm using a Spotify Audio Features dataset.

Data

The dataset for our project "Songs mood Classification" is taken from Kaggle (kaggle). It consists of 32k+ songs features and classifies them into happy and sad moods. There are nine features in the dataset such as loudness, danceability, etc. Also, happy and sad are the two labels that we considered for the ML model. We evaluated our production model on two datasets: the kaggle dataset with only two classes and 32k+ records, and the other manually gathered from a Spotify playlist with eight classes and 3k records, with the ground truth being the Spotify playlist name.

Data Graph (Features Correlation Graph)

The below figure shows the correlation between different songs features in the form of a heat map. We could infer that the energy and loudness are highly proportional and correlated. Also, Valence is equally related to danceability and energy.



**All the figures and accuracies are obtained from Dataset with 2 labels, Happy and Sad

Classic ML Model (RFC)

We have built and run several classical machine learning models so that we may find the best model among them to compare it to deep learning model. The best performing model out of all classical machine learning models was RFC or Random Forest Classifier. It had a training accuracy of 50% and testing accuracy of 45%.

For the hyper-parameter tuning, we performed GridSearch to arrive at the best possible combination of hyperparameters. We performed GridSearchCV on all the classical machine learning models and compared their accuracies to arrive at a conclusion of which is the best performing model.

Baseline ANN Model

A baseline model helps in giving a direction for the feature and hyperparameters engineering. We have divided the dataset into training and testing with a ratio of 4:1. We have nine features and evaluated the performance on both the datasets.

We started the model with one hidden layer having Relu activation and the output layer with the SoftMax activation along with sparse cross entropy loss. Furthermore, initially the model is trained with the Adam optimizer with a 0.001 learning rate. The Baseline model training accuracy achieved is around 62% and testing accuracy around 64%.

Production ANN Model

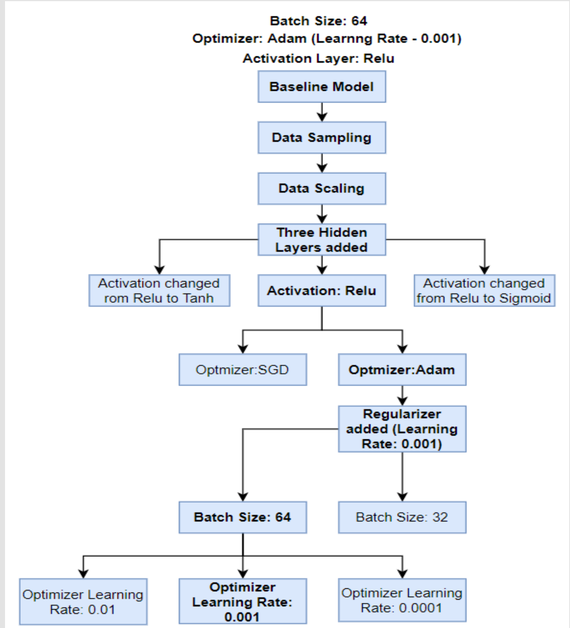
Once the baseline model is trained and tested, the model is improved by training the model again with different combinations of features and hyperparameters. First, we implemented the data sampling and scaling to balance the testing and training dataset and to remove class imbalance. Then, we added three more hidden layers to improve the accuracy. The three additions improved the model accuracy from 62% to 73%.

Next, we trained the model with different activation functions, i.e., Relu, tanh and Sigmoid. Furthermore, we chose different optimizers (Adam and SGD), batch size, optimizers learning rates to improve the accuracy of the model.

Lastly, we added the Regularizer to train and test the model which improved the accuracy to 75%.

Production ANN Model Architecture

The below figure shows how we built and developed the production ANN model from the baseline model where model is trained by adding sampling, scaling, hidden layers, activation functions, optimizer and the Regularizer.



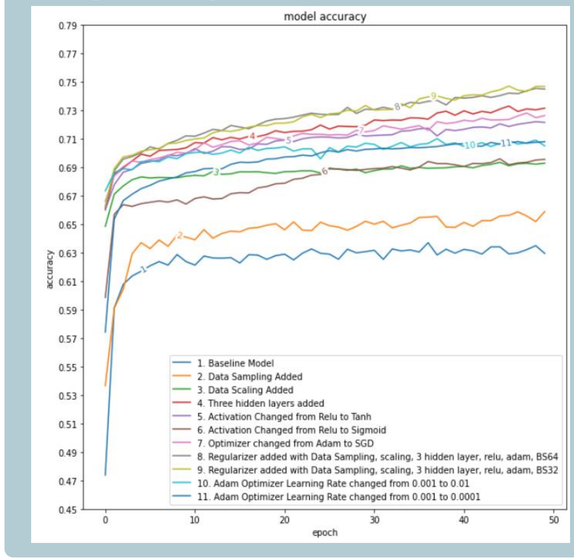
Misclassification

Upon random selection and analysis of 10 happy and 10 sad songs which were misclassified by the model, we found the following:

- The songs which were misclassified as happy by the model had very high acousticness, danceability and energy values, but when listening to these actual songs, we found out that the lyrics were sad. For example, the sad song was actually a Jazz or Metal song.
- The songs which were misclassified as sad by the model had low acousticness, danceability and energy and liveness values, but the lyrics were happy or motivational. The music was soothing or concentration music, which lacked energy and danceability.
- From this we can infer that the classification in Spotify was done by also considering the songs' lyrics, whereas our model focusses more on the audio features.

Comparative Analysis: ANN Model

The below chart displays the comparative study of variation in accuracies when tested at different steps during the build up of our final architectures.



ANN Model Analysis

- Since we see unbalanced datasets, the model will have hard time learning the decision boundaries between the classes. Hence, we are using SMOTE technique (SMOTETomek), i.e., data sampling to balance the records across the labels.
- 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.
- In order to make the data non-linearly separable, we added three hidden layers. Increasing the number of hidden layers is one of the ways to improve the performance of the model. If the number of hidden layers are increased beyond a certain limit or beyond what is needed then the performance of the model will decrease on the test set.
- Relu solves the problem of vanishing gradients and allowed neural networks to be of large sizes, hence accuracy is better than tanh and sigmoid.
- 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.