ISSN No:-2456-2165

Music Genre Detection using Machine Learning Algorithms

Karan Rathi[1]
Department of computer
Science and Engineering, Sharda University, SET
Greater Noida, UP (India)

Manas Bisht[2]
Department of Computer
Science and Engineering, Sharda University, SET
Greater Noida, UP (India)

Abstract:- Music genre classification is one example of content-based analysis of music signals. Historically, human-engineered features were employed to automate this process, and in the 10-genre classification, 61% accuracy was attained. Even yet, it falls short of the 70% accuracy that humans are capable of in the identical activity. Here, we suggest a novel approach that combines understanding of the neurophysiology of the auditory system with research on human perception in the classification of musical genres. The technique involves training a straightforward convolutional neural network (CNN) to categorise a brief portion of the music input. The genre of the song is then identified by breaking it up into manageable chunks and combining CNN's predictions from each individual chunk. The filters learned in the CNN match the Spectro temporal receptual field (STRF) in humans, and after training, this approach reaches human-level (70%) accuracy.

I. INTRODUCTION

Music plays a very important role in people's lives. Music brings like-minded people together and is the glue that holds communities together. A music genre is a category or classification of music that shares common characteristics such as musical style, instrumentation, rhythm, melody, and cultural and historical context. Examples of music genres include rock, pop, hip-hop, classical, jazz, blues, country, electronic, folk, and many others. Each genre is defined by a set of conventions that distinguish it from other genres and often has a dedicated fan base and industry infrastructure. The boundaries between genres can sometimes be blurred, and new genres can emerge through a fusion of existing ones or by incorporating elements of different styles. Music genre detection is the process of automatically identifying the genre of a piece of music using algorithms and machine learning techniques. The goal of music genre detection is to classify a piece of music into one or more predefined categories based on its acoustic features, such as timbre, rhythm, harmony, and melody. Music genre detection is used in various applications such as music recommendation systems, music streaming platforms, and content-based music retrieval systems. The process typically involves analyzing the audio signal using signal processing techniques to extract relevant features, which are then fed into machine learning models trained on labeled datasets. The models learn to recognize patterns and associations between the extracted features and the corresponding genre labels, which allows them to classify new, unlabeled pieces of music into the appropriate genre category. The genres of music that different communities write or even just listen to can be used to identify them. Different groups and communities listen to various types of music. The music's genre is a key characteristic that distinguishes it from other types of music. A regular person cannot recognize the genre of the music right away after listening to it. But because the distinctions between many genres of music can be hazy, classifying them is a particularly challenging job. For instance, in a test using a 10-way forced choice problem, college students were able to classify the music 70% accurately after hearing it for just 3 seconds, and the accuracy remained constant with longer music [1]. Additionally, the amount of tagged data is sometimes significantly lower than the data's dimension. For instance, even though the GTZAN dataset used in this work only has 1000 audio tracks, each audio track is 30 seconds long and has a sample rate of 22,050 Hz.

II. LITRATURE REVIEW

Numerous research papers on the classification of musical genres have extensively employed this kind of methodology. Multiple spectrograms obtained from audio recordings are used as inputs for CNN, and their patterns are extracted into a 2D convolutional layer with the appropriate filter and kernel sizes [9]. The spectrogram is mentioned in CNN because the model is good at identifying picture details [8]. Lau proposed applying the Convolutional Neural Network (CNN) model using a preprocessed GTZAN dataset. Each song's extracted Mel-Frequency Cepstrum Coefficient (MFCC) spectrogram was included in the dataset. Additionally, the feature descriptions for the audio excerpts in 3 seconds and 30 seconds were included in a separate.csv file [8]. Then, using Keras, he created a CNN architecture with 5 convolutional blocks. Each block contained a convolutional layer with a 3x3 filter and a 1x1 stride, a max pooling with a 2x2 windows size and a 2x2 stride, and a Rectifying Linear Unit (ReLU) function to display the probabilities for 10 music genres; the genre with the highest probability was picked as the input's classification label [8]. Twenty MFCCs were trained on 30-second and 3-second pieces of music, three CNN models were built on spectrograms, and a classification test was run on the test sets following training [8]. As Lau noted, there was a problem with the training datasets because the 3-second dataset did not match the number of genres in the sample Nevertheless, some genres featured fewer or more

samples than the standard (1000) [10]. The Short-term Fourier Transform (STFT) spectrograms, which are composed of different sequences of spectrogram vectors across time, were used by Yu et al to establish the CNN method [10]. In their paper, two datasets were mentioned: Extended Ballroom and GTZAN. Yu et al. separated each song from both datasets into 18 smaller parts in 3 seconds with 50% overlaps, increasing the data size set for each genre label by 18 times over the original [10]. The STFT spectrograms were examined with an analysis size of 513x128 and the train-validate-test ratio was 8:1:1 [10]. In order to capture discrete audio properties reflected in the STFT spectrograms and lessen source loss, pooling kernels and convolution filters were designed in small sizes in the first few layers of the CNN model [10]. Athulya and Sindhu came up with the idea of building a 2D Convolutional Neural Network (CNN). They extracted the audio samples from the GTZAN dataset into several types of spectrograms using the Librosa tool. These spectrograms served as binary inputs for the 2D CNN model developed with the Keras framework. The layers were also created using the TensorFlow framework [6]. Displayed was a 2D convolutional layer using input measurements of 128x128x1. The inputs to the max-pooling layer, which would operate a matrix half the size of the input layer, were represented by a 2D NumPy array [6]. The overall number of convolutional layers was 5, with a max-pooling layer, a stride of 2, and a 2x2 kernel size. Next, the output from each layer would be inserted into a fully linked layer that also had inputs in the form of a flattened and shrinking matrix size [6]. The SoftMax function, which was included at the end of the output layer, produced the probability output. The architecture achieved 94% accuracy. Similar to this, Nandy and Agrawal suggested a 2D CNN with a 1D kernel based on spectrograms generated from audio snippets in the Free Music Archive (FMA) dataset. The model produced an output of a 5000-length vector from an input dimension of 500x1500. Convolution layer blocks, a batch normalization layer, an activation layer, and, if practical, a max-pooling layer were all included in the construction of the CNN. The 2D CNN model was trained, validated, and tested 80:10:10 times, with a dropout parameter of 0.5 [1]. The model beat previous models from comparable research articles, performing with an accuracy rate of 76.2% and a logloss rate of 0.7543. An F1-Score greater than 0.7 indicated that the model was increasingly performing well at categorizing musical genres.

III. METHODOLOGY

A. *Pre-Processing*

We used data sets from the FMA medium, which are 25000 songs from 8 different genres that have been compressed to 30 seconds apiece. We have divided the unsorted datasets into only four genres based on the meta data, namely: Hip-Hop, rock, pop, and folk One audio clip was fed into our programme, and its spectral centroid, spectral bandwidth, spectral roll off, MFCC (Mel-frequency Cepstral Coefficients), Zero crossing rate, and RMSE (Root mean Square Energy) properties were extracted from it and saved into a new CSV file. Thus, obtaining a set of tagged data.

B. *Machine Learning Tecniques*

We used several models such as KNN, SVM, Naïve Bayes, Decision tree and NN.

- *KNN (K-Nearest Neighbours):*

This machine learning method and algorithm can be applied to both classification and regression tasks. K-Nearest Neighbours uses the labels of a predetermined number of data points to create a prediction about the class that the target data point belongs to. We utilised K-Nearest Neighbours (KNN), a conceptually straightforward yet incredibly effective technique, to train our model. Avoid combining SI and CGS units, such as magnetic field in oersteds and current in amperes. Due to the fact that equations do not balance dimensionally, this frequently causes confusion. If mixed units must be used, be sure to specify them for each quantity you include in an equation.

- *SVM (Support Vector Machine):*

This approach for supervised learning is utilised for both regression and classification. Finding a hyperplane in an N-dimensional space that clearly classifies data points is the basic goal of SVM.

- *Naïve Bayes:*

This model that makes predictions based on probability, and is an algorithm based on the idea of the Bayes theorem. It can also be used to solve classification problems.

- *Decision Tree:*

The non-parametric supervised learning approach used for classification and regression applications is the decision tree. It is organised hierarchically, with a root node, branches, internal nodes, and leaf nodes.

- *NN (Neural Networks):*

The foundation of deep learning algorithms is a subset of machine learning known as artificial neural networks (ANNs), often known as ANNs. Their design is influenced by the structure of the human brain, replicating how synapses are sent and received in the brain.

C. *Data Set*

We have used the data set called FMA medium which consists of **fma_medium.zip**: 25,000 tracks of 30s, 16 unbalanced genres (22 GiB). FMA stands for free music archive.

IV. EXPERIMENTAL RESULT

- *SVM*:

Table 1 SVM

Kernel	Acc	Pre	F1_score	Recall
rbf	0.729831	0.670305	0.609726	0.633856
poly	0.696998	0.655919	0.607322	0.60915

ISSN No:-2456-2165

➢ *SVM with Oversampling Techniques:*

Table 2 SVM with Oversampling Techniques

Kernel	Acc	Pre	F1_score	Recall	Sampler
rbf	0.745505	0.74467	0.740405	0.745175	SMOTE()
rbf	0.755878	0.753937	0.749485	0.755273	RandomOverSampler()
rbf	0.751037	0.749707	0.744337	0.750584	BorderlineSMOTE()
rbf	0.762102	0.763605	0.758042	0.761912	SVMSMOTE()
rbf	0.767956	0.767819	0.768145	0.769199	KMeansSMOTE(cluster_balance_threshold=0.1)
rbf	0.74537	0.746752	0.747558	0.748715	ADASYN(sampling_strategy='minority')
rbf	0.799447	0.821791	0.797449	0.799138	SMOTEN()

➢ *SVM with Undersampling Techniques:*

Table 2 SVM with Undersampling Techniques

Kernel	Acc	Pre	F1_score	Recall	Sampler
rbf	0.655684	0.652964	0.621252	0.653418	RandomUnderSampler()
rbf	0.558366	0.569202	0.550723	0.581744	CondensedNearestNeighbour()
rbf	0.686985	0.679718	0.66926	0.682761	NearMiss()
rbf	0.710049	0.713361	0.698142	0.707787	NearMiss(version=2)
rbf	0.589744	0.569426	0.559643	0.582002	NearMiss(version=3)
rbf	0.79878	0.783764	0.711256	0.738342	AllKNN()
rbf	0.617792	0.603224	0.599583	0.618626	ClusterCentroids()
rbf	0.838596	0.843487	0.827031	0.825887	EditedNearestNeighbours()
rbf	0.8	0.769459	0.713965	0.74829	NeighbourhoodCleaningRule()
rbf	0.719333	0.707028	0.612442	0.6415	OneSidedSelection()
rbf	0.863636	0.874934	0.840079	0.856448	InstanceHardnessThreshold()
rbf	0.838596	0.843487	0.827031	0.825887	RepeatedEditedNearestNeighbours()
rbf	0.734115	0.695274	0.623297	0.651841	TomekLinks()

➢ *KNN:*

Table 3 KNN

Neighbors	Acc	Pre	F1_score	Recall
1	0.637899	0.591337	0.591606	0.592258
2	0.63227	0.586035	0.58084	0.600804
3	0.663227	0.601234	0.600911	0.613214
4	0.653846	0.590138	0.592964	0.602144
5	0.662289	0.595443	0.59295	0.603027
6	0.664165	0.587951	0.586162	0.599344
7	0.670732	0.587503	0.583751	0.598749
8	0.67167	0.590921	0.586282	0.59977
9	0.665103	0.592297	0.58214	0.593952
10	0.672608	0.598224	0.584009	0.597841

➢ *KNN with Oversampling Techniques:*

Table 4 KNN with Oversampling Techniques

Neighbors	Acc	Pre	F1_score	Recall	Sampler
1	0.809129	0.812681	0.802575	0.803119	SMOTE()
1	0.806362	0.805534	0.802554	0.80244	RandomOverSampler()
1	0.810512	0.815597	0.804129	0.804392	BorderlineSMOTE()
1	0.793914	0.795273	0.788868	0.78886	SVMSMOTE()
1	0.787837	0.791687	0.785151	0.784242	KMeansSMOTE(cluster_balance_threshold=0.1)
1	0.74537	0.77089	0.73543	0.728385	ADASYN(sampling_strategy='minority')
9	0.79184	0.798838	0.788835	0.792201	SMOTEN()

➢ *KNN with Undersampling Techniques:*

Table 5 KNN with Undersampling Techniques

Neighbors	Acc	Pre	F1_score	Recall
10	0.640857	0.635161	0.63382	0.6345
10	0.395712	0.418999	0.401761	0.403933
9	0.685338	0.67878	0.659999	0.67032

➢ *TREE:*

Table 6 TREE

Criterion	Splitter	Acc	Pre	F1_score	Recall
gini	best	0.553471	0.497942	0.49952	0.50147
gini	random	0.560976	0.515978	0.515528	0.515315
entropy	best	0.54409	0.499232	0.500121	0.501359
entropy	random	0.536585	0.500461	0.498927	0.497801

➢ *TREE with Oversampling Techniques:*

Table 7 TREE with Oversampling Techniques

Criterion	Splitter	Acc	Pre	F1_score	Recall	Sampler
entropy	best	0.627939	0.625449	0.625452	0.626675	SMOTE()
entropy	random	0.630705	0.627423	0.626938	0.629347	SMOTE()
entropy	best	0.731674	0.73034	0.725539	0.730583	RandomOverSampler()
entropy	random	0.713001	0.710339	0.706784	0.711267	RandomOverSampler()
entropy	best	0.612725	0.611067	0.61092	0.612001	BorderlineSMOTE()
entropy	random	0.618949	0.615937	0.616585	0.618105	BorderlineSMOTE()
gini	best	0.615491	0.613362	0.613819	0.614844	SVMSMOTE()
gini	random	0.636238	0.635464	0.634589	0.635416	SVMSMOTE()
gini	best	0.639503	0.641571	0.640642	0.641131	KMeansSMOTE(cluster_balance_threshold=0.1)
gini	random	0.625691	0.625414	0.626336	0.628582	KMeansSMOTE(cluster_balance_threshold=0.1)
entropy	best	0.594907	0.599183	0.598668	0.600293	ADASYN(sampling_strategy='minority')
entropy	random	0.576389	0.587134	0.582343	0.581311	ADASYN(sampling_strategy='minority')
entropy	best	0.657676	0.656067	0.656327	0.657264	SMOTEN()
entropy	random	0.655602	0.655942	0.655029	0.654833	SMOTEN()

➢ *TREE with Undersampling Techniques:*

Table 8 TREE with Undersampling Techniques

Criterion	Splitter	Acc	Pre	F1_score	Recall	Sampler
entropy	best	0.518946	0.519458	0.520445	0.521913	RandomUnderSampler()
entropy	random	0.479407	0.478521	0.4793	0.480302	RandomUnderSampler()
entropy	best	0.392996	0.401074	0.396744	0.393597	CondensedNearestNeighbour()
entropy	random	0.365759	0.371371	0.366525	0.363827	CondensedNearestNeighbour()
gini	best	0.553542	0.557621	0.552564	0.549281	NearMiss()
gini	random	0.5486	0.550175	0.54933	0.548624	NearMiss()
entropy	best	0.530478	0.536263	0.532219	0.529644	NearMiss(version=2)
entropy	random	0.482702	0.483314	0.481999	0.484062	NearMiss(version=2)
gini	best	0.43956	0.437988	0.435287	0.433024	NearMiss(version=3)
gini	random	0.437729	0.435544	0.433654	0.434265	NearMiss(version=3)
gini	best	0.664634	0.640913	0.638381	0.636145	AllKNN()
gini	random	0.618293	0.592288	0.591141	0.590358	AllKNN()
entropy	best	0.4514	0.447941	0.448931	0.453587	ClusterCentroids()
entropy	random	0.395387	0.39946	0.39775	0.396775	ClusterCentroids()
entropy	best	0.677193	0.685906	0.681607	0.678929	EditedNearestNeighbours()
entropy	random	0.624561	0.631493	0.626196	0.62233	EditedNearestNeighbours()
gini	best	0.643787	0.619108	0.61944	0.622804	NeighbourhoodCleaningRule()
gini	random	0.642604	0.611317	0.613046	0.618897	NeighbourhoodCleaningRule()
entropy	best	0.572978	0.528531	0.530101	0.531971	OneSidedSelection()
entropy	random	0.527613	0.489196	0.488243	0.487685	OneSidedSelection()
entropy	best	0.754902	0.747541	0.74659	0.746123	InstanceHardnessThreshold()
entropy	random	0.740196	0.730401	0.729952	0.729724	InstanceHardnessThreshold()
entropy	best	0.684211	0.69034	0.687178	0.686358	RepeatedEditedNearestNeighbours()
entropy	random	0.65614	0.659947	0.652301	0.645925	RepeatedEditedNearestNeighbours()
gini	best	0.580645	0.550362	0.547944	0.546856	TomekLinks()
gini	random	0.55523	0.52485	0.522069	0.520656	TomekLinks()

➢ *NAÏVE BAYES:*

Table 9 NAÏVE BAYES

NB	Acc	Pre	F1_score	Recall
GaussianN	0.624765	0.55081	0.5497	0.564986
Compleme	0.56379	0.441872	0.44776	0.510773
BernoulliN	0.333021	0.083255	0.124912	0.25

➢ *NAÏVE BAYES with Oversampling Techniques:*

Table 10 NAÏVE BAYES with Oversampling Techniques

NB	Acc	Pre	F1_score	Recall	Sampler
GaussianN	0.545643	0.508588	0.514871	0.541935	SMOTE()
GaussianN	0.545643	0.508588	0.514871	0.541935	RandomOverSampler()
GaussianN	0.545643	0.508588	0.514871	0.541935	BorderlineSMOTE()
GaussianN	0.545643	0.508588	0.514871	0.541935	SVMSMOTE()
GaussianN	0.545643	0.508588	0.514871	0.541935	ADASYN()
GaussianN	0.545643	0.508588	0.514871	0.541935	SMOTEN()

➢ *NAÏVE BAYES with Undersampling Techniques:*

Table 11 NAÏVE BAYES with Undersampling Techniques

NB	Acc	Pre	F1_score	Recall	Sampler
GaussianN	0.565007	0.532474	0.535836	0.556526	RandomUnderSampler()
GaussianN	0.577982	0.410988	0.416725	0.448746	CondensedNearestNeighbour()
GaussianN	0.531561	0.54066	0.523722	0.526333	NearMiss()
GaussianN	0.538206	0.54845	0.533789	0.541295	NearMiss(version=2)
GaussianN	0.458333	0.411	0.368815	0.40311	NearMiss(version=3)
Compleme	0.654545	0.305423	0.311184	0.317647	AllKNN()

➢ *Random Forest:*

Table 12 Random Forest

Criterion	Splitter	Acc	Pre	F1_score	Recall
gini	100	0.697936	0.65056	0.601014	0.620694
gini	200	0.703565	0.68069	0.617106	0.631219
gini	300	0.705441	0.689496	0.620962	0.633594
gini	400	0.709193	0.696107	0.623671	0.636403
gini	500	0.708255	0.687861	0.620429	0.634207
gini	600	0.706379	0.686563	0.618993	0.632803
gini	700	0.709193	0.6887	0.621062	0.634907
gini	800	0.707317	0.683413	0.619912	0.634028
gini	900	0.707317	0.683581	0.619952	0.634028
entropy	100	0.696998	0.674651	0.60599	0.623394
entropy	200	0.696998	0.660053	0.602065	0.621465
entropy	300	0.696998	0.682201	0.60389	0.621385
entropy	400	0.698874	0.686061	0.607411	0.624273
entropy	500	0.698874	0.668456	0.605225	0.623307
entropy	600	0.696998	0.673087	0.601557	0.620415
entropy	700	0.69606	0.672353	0.601322	0.620757
entropy	800	0.697936	0.677736	0.605042	0.623128
entropy	900	0.697936	0.672421	0.604885	0.623128

➢ *Random Forest with Oversampling Techniques:*

Table 13 Random Forest with Oversampling Techniques

Criterion	Splitter	Acc	Pre	F1_score	Recall
gini	800	0.778008	0.776491	0.776455	0.777311
entropy	800	0.784232	0.782807	0.782665	0.78356
gini	900	0.826418	0.826478	0.825545	0.82604
entropy	800	0.824343	0.824812	0.82349	0.823906
gini	700	0.785615	0.784106	0.783409	0.784772
entropy	900	0.786307	0.785257	0.78409	0.785417
gini	800	0.782849	0.783151	0.780814	0.781759
entropy	900	0.784924	0.786638	0.783283	0.783922
gini	800	0.786602	0.795048	0.785495	0.784966
entropy	700	0.790055	0.798784	0.789301	0.788547
gini	400	0.766334	0.767424	0.766181	0.766333
entropy	900	0.767102	0.770926	0.768359	0.76747
gini	800	0.784232	0.801157	0.783942	0.782943
entropy	400	0.786307	0.803378	0.785555	0.784884

➢ *Random Forest with Undersampling Techniques:*

Table 14 Random Forest with Undersampling Techniques

Criterion	Splitter	Acc	Pre	F1_score	Recall	Sampler
gini	800	0.60461	0.58568	0.58849	0.60355	RandomUnderSampler()
entropy	900	0.61285	0.59626	0.5959	0.61173	RandomUnderSampler()
gini	900	0.51961	0.52605	0.52134	0.52228	CondensedNearestNeighbour()
entropy	100	0.52157	0.52679	0.5236	0.52491	CondensedNearestNeighbour()
gini	900	0.70181	0.69526	0.69063	0.70047	NearMiss()
entropy	900	0.70511	0.70236	0.68882	0.7039	NearMiss()
gini	900	0.68863	0.68767	0.68015	0.68929	NearMiss(version=2)
entropy	600	0.69358	0.69686	0.68283	0.69436	NearMiss(version=2)
gini	800	0.56514	0.55397	0.5533	0.56405	NearMiss(version=3)
entropy	900	0.55963	0.55031	0.54566	0.55769	NearMiss(version=3)
gini	900	0.78335	0.74691	0.70508	0.71847	AllKNN()
entropy	800	0.78091	0.7558	0.70185	0.71669	AllKNN()
gini	900	0.53213	0.51348	0.51876	0.52972	ClusterCentroids()
entropy	100	0.56013	0.53968	0.54443	0.55836	ClusterCentroids()
gini	800	0.7993	0.79563	0.79399	0.80821	EditedNearestNeighbours()
entropy	900	0.79578	0.79467	0.7898	0.80602	EditedNearestNeighbours()
gini	600	0.75924	0.6935	0.67827	0.7061	NeighbourhoodCleaningRule()
entropy	400	0.76281	0.72032	0.67597	0.70812	NeighbourhoodCleaningRule()
gini	200	0.70949	0.69316	0.63045	0.63547	OneSidedSelection()
entropy	600	0.70553	0.69857	0.61866	0.62889	OneSidedSelection()
gini	700	0.85714	0.86012	0.84558	0.86235	InstanceHardnessThreshold()
entropy	900	0.8474	0.85103	0.83331	0.85304	InstanceHardnessThreshold()
gini	800	0.7993	0.79563	0.79399	0.80821	RepeatedEditedNearestNeighbours()
entropy	700	0.79754	0.79603	0.79247	0.80777	RepeatedEditedNearestNeighbours()
gini	700	0.70207	0.68689	0.61393	0.63224	TomekLinks()
entropy	900	0.70207	0.71284	0.61057	0.63051	TomekLinks()

➢ *Oversampling and Undersampling Used:*

- *SYNTHETIC MINORITY OVERSAMPLING (SMOTE):* This statistical method is employed to evenly increase the number of examples in your dataset. SMOTE creates new instances from inputs of minority cases that already exist.

- *SMOTE-NC:* It is used to generate synthetic data to oversample a minority target class in an imbalanced dataset.
- *ADASYN:* The major benefits of this technique, which creates synthetic data, are duplicating minority data and producing extra data for "harder to learn" examples.
- BORDERLINE-SMOTE: This algorithm classifies any minority observation as a noise point if all the neighbours are of majority class, and such an observation is ignored while creating synthetic data.
- *K-MEANS SMOTE:* is an oversampling method for class-imbalanced data. It aids classification by generating minority class samples in safe and crucial areas of the input space.
- SVM SMOTE: is a Variant of SMOTE algorithm which use an SVM algorithm to detect sample to use for generating new synthetic samples.
- *CLUSTER CENTROIDS:* is a method that undersamples the majority class by substituting the cluster centroid of a KMEANS algorithm for a cluster of majority sample locations.
- *CONDENSED NEAREST NEIGHBOUR:* Condensed nearest neighbour which is also known as the Hart algorithm is an algorithm designed to reduce the data set for k-NN classification. It selects the set of prototypes U from the training data, such that 1NN with U can classify the examples almost as accurately as 1NN does with the whole dataset.
- *EDITED NEASREST NEIGHBOUR (ENN):* This method works by finding the K-nearest neighbour of each observation first, then check whether the majority class from the observation's K-nearest neighbour is the same as the observation's class or not.
- *REPEATED EDITED NEASRESTNEIGHBOUR:* This method repeats the ENN algorithm several times; it under samples based on the repeated edited nearest neighbour.
- *AIIKNN:* It removes all examples from the dataset that were classified incorrectly.
- *INSTANCEHARDNESS THRESHOLD:* This is an under sampling method that was built to tackle imbalanced classifications.
- *NEARMISS:* It refers to a group of undersampling techniques that choose examples depending on how close majority class and minority class examples are to one another.
- *ONESIDEDSELECTION:* Condensed Nearest Neighbour (CNN) Rule and Tomek Links are two undersampling techniques that are combined to create One-Sided Selection, or OSS. The CNN approach is used to eliminate redundant examples from the interior of the density of the majority class, whereas the Tomek Links method is used to eliminate noisy examples on the class boundary.
- *RANDOMUNDERSAMPLER:* It undersamples the majority class(es) by randomly picking samples with or without replacements.
- *TOMEKLINKS:*Tomek links are pairs of instances of opposite classes who are their own nearest neighbours.
- *RANDOM OVERSAMPLER:* Machine learning uses the method of random oversampling to balance unbalanced datasets. One class contains significantly fewer examples than the other(s) in an unbalanced dataset. This may result in a biassed model that does not adequately represent the minority class. This is addressed by random oversampling, which duplicates examples from the minority class until the dataset is balanced.

V. FUTURE WORK

➢ *There are several future scopes that can be explored to improve its performance. Here are some possible approaches:*

- *Data Augmentation:*

One way to improve the performance of a genre detection model is to increase the size of the training dataset by generating additional examples from the existing ones through data augmentation techniques. For example, the audio signals can be randomly time-stretched, pitch-shifted, or filtered to create variations of the same piece of music that can help the model learn more robust representations of the genre features.

- *Feature Engineering:*

Another way to improve the performance of a genre detection model is to extract more informative features from the audio signals that capture the essential characteristics of each genre. This can be achieved by using more sophisticated signal processing techniques or by incorporating domain-specific knowledge about music theory and composition into the feature extraction process.

- *Ensemble Methods:*

Ensemble methods combine the outputs of multiple models to make more accurate predictions than any single model alone. By training multiple genre detection models with different architectures, hyperparameters, or training data, and then combining their outputs through voting, averaging, or stacking, we can leverage the diversity of the models' predictions to improve the overall accuracy of the ensemble.

- *Transfer Learning:*

Transfer learning involves reusing pre-trained models that were originally trained on large datasets for related tasks to improve the performance of a new model with limited training data. By fine-tuning a pre-trained model on a smaller genre detection dataset, we can leverage the pre-existing knowledge captured by the model to improve its accuracy on the target task.

- *Hybrid Approaches:*

Hybrid approaches combine multiple techniques from the above methods to create more sophisticated genre detection models. For example, a hybrid model could use a pre-trained deep learning model for feature extraction, followed by a support vector machine (SVM) classifier trained on augmented data, and then an ensemble method to

combine the predictions of multiple SVM models with different hyperparameters.

VI. CONCLUSION

In this paper we propose the KNN algorithm with EditedNearestNeighbor undersampling and only one neighbour to be far more accurate than the remaining algorithms along with their over and under sampling. Due to our limited computational resources and time we were not able to execute neural network on our dataset.

REFERENCES

[1]. Chatziagapi, A., Paraskevopoulos, G., Sgouropoulos, D., Pantazopoulos, G., Nikandrou, M., Giannakopoulos, T., ... & Narayanan, S. (2019, September). Data Augmentation Using GANs for Speech Emotion Recognition. In Interspeech (pp. 171-175).

[2]. Biswas, R., & Ghattamaraju, N. (2019). An effective analysis of deep learning based approaches for audio based feature extraction and its visualization. Multimedia Tools and Applications, 78, 23949-23972.

[3]. Lu, Y. C., Wu, C. W., Lerch, A., & Lu, C. T. (2016, August). Automatic Outlier Detection in Music Genre Datasets. In ISMIR (pp. 101-107).

[4]. Fell, M., & Sporleder, C. (2014, August). Lyrics-based analysis and classification of music. In Proceedings of COLING 2014, the 25th international conference on computational linguistics: Technical papers (pp. 620-631).

[5]. Van Mieghem, L. C. F. (2020). Music Genre Detection: with Neural Networks.