Music Genre Classification Using KNN, SVM, and CNN

Burcu Arslan, *Department of Computer Science, Ozyegin University*
Aleyna Nur Ölmezcan, *Department of Computer Science, Ozyegin University*
Emin Sadikhov, *Department of Computer Science, Ozyegin University*
Abdullah Saydemir, *Department of Computer Science, Ozyegin University*
Muhammed Esad Simitçioğlu, *Department of Computer Science, Ozyegin University*

***Abstract*—Music classification is useful in terms of indexing and retrieval of music, although it is a difficult process to perform systematically and consistently due to the subjective nature of music genres. To come up with an efficient classification technique, machine learning methods utilizing lyrics, audio sources, images and a combination of those metrics are proposed. In this article, we analyze classification of music genres using Support Vector Machines (SVM), Convolutional Neural Networks (CNN) and K-Nearest Neighbor (KNN) algorithms in the GTZAN dataset which is considered as the benchmark. We also provide results obtained from this research alongside comparison to those of the previous works in the literature and brief discussion of comparative advantages.**

***Index Terms*—Classification, music, music genre, Convolutional Neural Network (CNN), Support Vector Machines (SVM), K-Nearest Neighbor (KNN).**

I. Introduction

MUSIC features can provide high-level information about the content of the music. These are often related to genre, mood, instrumentation, and the like. It is therefore considered as a combination of tasks such as species classification and instrument recognition [2]. One of the most common features to consider when recommending a certain music track to a user and retrieving music is genre. One approach to categorize and arrange songs is by their genre, which is defined by different features of the music such as rhythmic structure, harmonic content, and instrumentation. Music genre classification studies are carried out because manually listening and classifying music causes a waste of time. Music classification software can make it simpler to spot important details such as trends, popular genres, and artists. As the amount of music released on a daily basis continues to rise, particularly on internet platforms like Soundcloud and Spotify, the need for accurate meta-data for database management and search/storage purposes rises in tandem. The ability to instantly categorize songs in any given playlist or library by genre is an important functionality for any music streaming/purchasing service, and the statistical analysis capacity that correct and complete labeling of music and audio provides is essentially limitless. On the same dataset, this article tested classification using three different algorithms: Support Vector Machines, Convolutional Neural Networks, and K-Nearest Neighbor, and attempted to guess the genre from ten common music genres. While analyzing the classification methods in our study, we obtained the most successful accuracy rate in Convolutional Neural Networks (CNN).

II. Related Work

In music classification, studies generally employ audio sources [3]. Earlier studies employed constant features such as Zero Crossing Rate or spectral decrease. On the other hand, newer ones prefer using graphical representations to the problem. Such inputs are derived using spectrograms (e.g. mel spectrogram, tonnetz etc.). These inputs are fed to popular classification models such as Convolutional Neural Networks or Recurrent Neural Networks. Some of these studies can be listed as follows. Table I shows a summary about all the previous work mentioned here.

Chen and Stevan [4] propose music genre classification method (ATMGCM) for musical genre classification field. Through experiments, this analysis shows that ATMGCM algorithm is effective at classifying genres. In the final analysis, only 10%-15% of GTZAN dataset needs to be labeled to achieve good results.

Puppala et al. [5] used the GTZAN dataset like our work. They used the Mel Frequency Cepstral Constant (MFCC) as a function vector to utilize sound sample. They extracted the feature vector so that the structured framework divides music into various genres. Utilizing CNN, the project has an accuracy level of 97% for training and 74% for testing.

To give examples of the use of Support Vector Machines in literature of music genre classification, Chaudary et al. [6] proposed a model that uses Empirical Mode Decomposition (EMD) to extract data and then chooses the first eight Intristic Mode Functions (IMF) before extracting the time and frequency domain features, next the linear Support Vector Machine is implemented to get an accuracy of 94%.

Sharma et al. [7] offers a classification method using acoustic properties of audio files using the GTZAN dataset like our work. They combine three different results obtained using SVM, Relevance Vector Machine (RVM) and Decision Trees as a classifier, using the sum rule, and obtain a hybrid result. In this relevant study, the maximum accuracy of 87% was achieved as a hybrid. The accuracies of the individual classifiers are shown in Table I.

In research from Nanni et al. [8], the accuracy score was found to be at 87%. Their model used acoustic and visual

data, by taking the features of the sound while creating a visual representation of the audio signal. CNN is also trained with these visual representations. The acoustic features are classified with SVM after being extracted. The results from the SVM and CNN are then combined to get a final classification.

In work from Xu et al. [9], a multi-layer classifier based on SVM is implemented, and in definite layers, the sound is classified into some genres according to the features of the beat spectrum, LPC-derived cepstrum, spectrum power, MFCCs and zero crossing rates. SVM is used in all layers and each has different hyperparameters. For 4 class classification, 93% accuracy was obtained.

Tamatija and Mahastama [10] classified twelve music genres from the songs they obtained using features such as Zero Crossing Rate, Average Energy and Silent Ratio, with KNN algorithm. They analyzed the results by experimenting with the number k in different ways, and their best result was 56%. Though the accuracy may seem lower, this study may be effective since there is too much beat in the music tracks used in the study.

Şimşekli [11] performed an automatic music genre classification study based on the bass line. MIDI dataset is used in this study. Reported classification accuracy is 84% with the k-NN classifiers and the added utility of the novel metric (PWED). We can interpret that the result obtained in this study is better than ours, as the bass lines are a sufficient and rich source of information in terms of defining the right genre, as stated in the cited article, and that bass lines are emphasized in that article.

Wu [12] proposed a double weighted algorithm where instead of using the traditional way of subtracting each feature pairs, attribute dependency κ is calculated and classification is done according to the occurrence of the sum of the weighted distance of samples that has $\kappa > 0$. This increased the classification accuracy by 5% (82 to 87) in two class classification and 13% (46 to 59) for 10 class classification on genres dataset compared to traditional KNN.

Cheng et al. [13] made use of a Convolutional Neural Network (CNN) model composed of five convolutional layers and ReLU as activation function to classify the music retrieved from the GTZAN dataset into 10 genres. After majority voting is applied, reported accuracy was %84.

In work from Zhang et al. [14] two models using CNN was proposed. To improve the performance of music genre prediction, in one of the models, max and average pooling were combined to provide better information to higher neural networks. In the other model, a method inspired on residual learning that uses shortcut connections to skip layers were implemented. The GTZAN dataset was used, and the highest accuracy achieved was 87.4%.

TABLE I
PREVIOUS WORK ON MUSIC GENRE CLASSIFICATION

Reference	Dataset	Model	Accuracy
Chen et. al [4]	GTZAN	SVM	0.76
		RF	0.75
		ATMGCM	0.82
Sharma et al. [7]	GTZAN	Hybrid	0.87
		SVM	0.84
		RVM	0.68
		Ensemble	0.79
Chaudary et al. [6]	GTZAN	SVM	0.94
Xu et al. [9]	Unspecified	SVM	0.93
Nanni et al. [8]	LMD	SVM & CNN	0.90
	GTZAN		0.87
	ISMIR 2004		0.85
Cheng et. al. [13]	GTZAN	CNN	0.84
Puppala et al. [5]	GTZAN	CNN	0.74
Zhang et al. [14]	GTZAN	CNN	0.84
Tamatija et al. [10]	Unspecified	KNN	0.56
Şimşekli [11]	MIDI	KNN	0.84
Wu et al. [12]	GTZAN	DW-KNN	0.59

III. METHODS

A. The Dataset

GTZAN [15] dataset that contains 30 second samples of 100 songs from 10 different genres, namely blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, rock, is downloaded. This dataset is specifically used because it is previously used in various number of studies. [4], [6], [7] Though, it contains misclassified true labels and repetitive instances, it is considered as the benchmark dataset in the field. Audios are loaded with sampling rate 22050 in mono format and 75 different features are extracted from the songs using Librosa [16] library. For some features that are not provided in the Librosa library, such as spectral spread or spectral decrease, PyAudioAnalysis [17] library is used. 46 more features are extracted using PyAudioAnalysis library, to make a sanity check across the libraries. Features provided in both of them are used (i.e. zero crossing rate, spectral centroid) and the results are compared. After comparison, features obtained with Librosa are preserved.

Features are normally represented as vectors (numpy arrays) with respect to time. Standard deviation and average of the values are obtained to represent each of the instances. Detailed view of the features used and the representations can be obtained from Table II.

B. Software

Code is mainly written in Python. Python libraries that are used in this research are Librosa [16], PyAudioAnalysis [17], scikit-learn [18], PyTorch [19], Keras [20]. Implementation and resulting log files are available through GitHub[1].

C. Models

Support Vector Machines (SVM), Convolutional Neural Networks (CNN) and K-Nearest Neighbor (KNN) algorithms were implemented for the classification through music genres for different kinds of songs.

[1]https://github.com/aleynaolmezcan/CS-454-Project

TABLE II
FEATURES EXTRACTED FROM DATASET

Librosa Library		
Feature Name	**Representation**	**Dimension**
Spectral Rollof	STD + Mean	2
Spectral Centroid	STD + Mean	2
Spectral Bandwidth	STD + Mean	2
Spectral Flatness	STD + Mean	2
Spectral Contrast	STD + Mean	2
Tempogram	STD + Mean	2
Tempo	Mean	1
Root Mean Square	STD + Mean	2
Zero Crossing Rate	STD + Mean	2
Chroma Short-Time Fourier Transform	STD + Mean	2
Chroma Energy Normalized	STD + Mean	2
Constant-Q Transform Chromagram	STD + Mean	2
Tonnetz	STD + Mean	12
Mel-Frequency Cepstral Coefficients	STD + Mean	40
Total		75
PyAudioAnalysis Library (Sanity Check)		
Zero Crossing Rate	STD + Mean	2
Spectral Centroid	STD + Mean	2
Spectral Rollof	STD + Mean	2
Mel-Frequency Cepstral Coefficients	STD + Mean	40
Total		46

1) Support Vector Machines: The goal of the Support Vector Machine is to find a hyperplane that separates the positive and negative data points with the greatest possible margin [21]. The given data is transformed using the kernel technique, and the optimal boundary between various events is determined by generating a hyperplane for each event or class based on these changes. While building the hyperplane, the maximum possible distance between the hyperplane and support vectors is maintained [6].

SVM was implemented with the help of Scikit-learn library [18]. The split of training to testing data was with a ratio of 80/20 as seen in the work of Sharma et al. [22] which also uses SVM for classification. The data was normalized with StandardScaler and MinMaxScaler of Scikit-learn library. The optimal hyperparameters for the model were retrieved with GridSearchCV with a three-fold cross-validation, and the optimal kernel was selected as Radial Basis Function (RBF) thus RBF kernel was implemented. The results for the accuracy, balanced accuracy, F1 Score and Precision Score were the same with the shuffling parameter set to True on train_test_split function of Scikit-learn when the code was executed for 10 different iterations.

2) Convolutional Neural Networks: Convolutional Neural Net, or ConvNet, is a type of deep artificial neural network that is used to evaluate visual information [23]. Convolutional neural networks are basically constituted of a convolutional layer, pooling layer and fully connected layer. The convolutional layer's concept is to get local characteristics of the audio or picture by sliding up and down consecutively through a window of a specific size which is convolutional kernel [13].

This was our most sophisticated model, with three convolation layers, each with its own max pool and regularization, feeding into three fully connected layers with ReLU activation, softmax output, and cross entropy loss. Convolution windows scan over the input data and output the sum of the elements within the window in this method. This is then sent into a max pool layer, which chooses the most significant element from another window. TensorFlow and Keras were used to do this.

3) K-Nearest Neighbor: K Nearest Neighbor is a non-parametric method that classifies a node based on the labels of the nearest K neighbors. Nearest neighbors are found by applying distance metrics to all pairs of vectors. After the distances to the neighbors are calculated, the label is predicted by getting the mod of the occurrences of labels of K nearest neighbors.

Mahalanobis distance, considers the correlation and variance among the features of the two vectors. Given two vectors of the same dimensions Mahalanobis distance is calculated using Equation 1.

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T \mathbf{C}^{-1} (\vec{x}-\vec{y})} \tag{1}$$

where C is the covariance matrix of the said vectors. If the covariance matrix is identity matrix, then the distance can be calculated using the standard euclidean distance formula, given by Equation 3.

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T (\vec{x}-\vec{y})} \tag{2}$$

Minkowski metric is also used in computations which is a generalization of the Euclidean distance and Manhattan distance. Manhattan distance between two vectors are calculated as

$$d(\vec{x},\vec{y}) = (\vec{x}-\vec{y})^T (\vec{x}-\vec{y}) \tag{3}$$

and Minkowski metric is calculated as

$$d(\vec{x},\vec{y}) = (\sum_{i=1}^{n} |\vec{x}_i - \vec{y}_i|^p)^{1/p} \tag{4}$$

When p is equal to 1, it can be easily seen that it transforms to Manhattan distance. Similary when p is 2, it is Euclidean distance.

K-NN algorithm was implemented with KNeighborsClassifier available in Scikit-learn library. Hyperparameters tested were distance metrics *(euclidean, minkowski, manhattan, chebyshev, mahalanobis)*, number of neighbors *(1 to 20)* and number of components PCA holds *(1 to all)*. These parameters were tested using a bash script (see appendices) rather than a Python library.

As the standard step in all of three models, inputs are scaled using MinMaxScaler. Input is shuffled and split into training, validation and test data. Ratios were 0.8, 0.1 and 0.1 to compare the result with those of the previous work.

D. Algorithms

1) GridSearchCV: GridSearchCV is an exhaustive search function from python Scikit-learn library. It generates hyper-parameter set candidates from the provided hyper-parameters by the user [24].

2) MinMaxScaler: MinMaxScaler from the python Scikit-learn library transforms features by scaling each feature to a given range [24]. Entries in the matrix are updated using Equation 5

$$X = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5}$$

where X_{min} is the minimum element vector and X_{max} is the maximum element vector for each row.

3) StandardScaler: StandardScaler is a data normalization method used to standardize the features of the data from python Scikit-learn library. It does so by removing the mean and scaling to unit variance [24].

$$z = \frac{x - \mu}{\sigma} \tag{6}$$

4) Principle Component Analysis: Principle Component Analysis (PCA) is an unsupervised algorithm used in dimensionality reduction. Aim is representing the original dataset with lesser dimensional vectors called Principal Components, while retaining the most information from the data.

E. Evaluation Metrics

We evaluated our models with accuracy, balanced accuracy, F1, precision and recall.

Since there is no need to impose various costs on different types of misclassification, most classification studies focus on average accuracy. The average accuracy across all folds is the most often used method for presenting cross-validation outcomes [25].

As this is a multiclass classification, the subset accuracy was computed. [24]

Balanced accuracy is the macro-average recall for each class, and for a balanced dataset this score is similar to accuracy. [24]

Recall is the ability of the classifier to successfully discover all the positive samples [1]. The sum of true positives across all classes is divided by the sum of true positives and false negatives across all classes to calculate recall score.

The sum of true positives across all classes is divided by the sum of true positives and false positives across all classes to get precision [1].

Lastly, F1 score is the harmonic mean of precision and recall scores.

$$F_1 = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{7}$$

IV. Results & Discussion

We will present and discuss our results in the following section. Table III shows the values of each evaluation metric for all three models.

TABLE III
BEST RESULTS FROM MODELS UTILIZED

Model	Accuracy	Balanced Accuracy	F1 Score	Precision	Recall
SVM	0.80	0.80	0.79	0.80	0.79
CNN	0.84	0.83	0.84	0.85	0.84
KNN	0.73	0.73	0.74	0.76	0.73

A. Support Vector Machine

According to Table 1, it can be seen that the model has a lower performance than some of the models in literature.

In the work of Chaudary et al. [6] the model also uses the GTZAN dataset, but uses and classifies into only 5 genres and EMD and IMF are implemented. to get an accuracy score of 94%. This difference in accuracy could be because of the use of EMD and IMF, along with the difference in the number of genres of the data used.

Previously mentioned work of Sharma et al. [7] uses GTZAN dataset and they merged three classifiers to reach maximum accuracy which was 87%. Even if the SVM classifier accuracy itself is 84% and still higher than our result, merging different classifiers could be beneficial for our study.

The approach of using both visual and acoustic data in work from Nanni et al. [8] could be the reason behind the difference in accuracy scores, along with the fusion of two different models. Also, in this model, the Latin Music Database and ISMIR 2004 was used along with GTZAN as the model's dataset and this difference in data could play a role in the accuracy scores.

B. Convolutional Neural Network

We try to maximize the accuracy with changing the batch size, epochs and dropout rate. Full results can be obtained from log files.

The best results obtained were with batch size 64, epoch 400 and dropout rate of 0.1. We expected higher accuracy from CNN, however, initially we assumed some problems in our implementation and ended up with 84% as our highest accuracy rate for CNN. According to Table 1, we can observe that CNN gave the same results with GTZAN dataset. Hereby, our assumption was not correct and there was no any problem in our implementation. Nevertheless, when we test it with images, it gives much greater performance. It seems CNN gives it's best result when the input is an image.

In the same dataset GTZAN, Puppala et al [5] for feature extraction they used MFCC is utilized for the sound sample. Following the extraction of the feature vector, they devised a framework for categorizing music into various genres. They achieve 74% accuracy which is way lower than our accuracy rate. Cheng et al. [13] convert the original audio file into their corresponding mel-spectrums. These mel spectrum is used in the CNN model for training They achieve 84% accuracy rate with this spectrum. Neverthless, our accuracy rate is same with their.

C. K-Nearest Neighbor

Best results of each metric for K-NN algorithm for different number of PCA components and num_neighbors can be seen from table Table IV. These values are outputs of 10 fold cross-validation. Full results can be obtained from log files.

The best results obtained with Euclidean metric when number of PCA components was 25 and k was 7. There were no significant difference in terms of the metrics considered. Initially, we thought that MinMaxScaler may not normalize

the variance. However, as it can be seen from the results this doubt was pointless. MinMaxScaler seems to provide good variance resolution.

Also, PCA was not very useful in dimensionality reduction since for best results number or features were > 25. This shows, features extracted were distinct and contributed to prediction. For constant K, one neighbors were enough for differentiation genres using Minkowski and Manhattan distances. Similarly, in all metrics considered, k was always <10.

Compared to Double Weighted KNN, we achieved promising results. For 10 class classification, our method provided 73% accuracy compared to Wu et al.'s [12] 59%. Our results were better than Tamajita et al.'s [10] work and was poorer than Şimşekli's [11] though the train test split ratios were the same. However, it is important to consider the use of different dataset and different number of genres in this classification problem.

TABLE IV
RESULTS OF K-NN FOR EACH DISTANCE METRIC

Metric	PCA Comp.	K	Accuracy	F1 Score	Precision	Recall
Chebysev	50	4	0.66	0.64	0.65	0.66
Euclidean	25	7	0.73	0.74	0.76	0.73
Mahalanobis	75	1	0.67	0.67	0.69	0.67
Manhattan	70	7	0.73	0.72	0.76	0.73
Minkowski	75	1	0.72	0.72	0.75	0.72

V. CONCLUSION

In this study, the classification of music data into 10 different genres from GTZAN dataset was implemented. Selected features were trained and tested using three different classification models namely Support Vector Machine, Convolutional Neural Networks and K-Nearest Neighbours. CNN performed the best out of the three models for all of the selected evaluation metrics with an accuracy of 84% whereas SVM got an accuracy of 79% and KNN 73%. Results show that the implementations provided comparable results to those of the previous work.

ACKNOWLEDGMENT

The authors would like to thank *Mahir Atmış* from Ozyegin University for his support during the semester.

REFERENCES

[1] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.
[2] B. Liang and M. Gu, "Music genre classification using transfer learning," in *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2020, pp. 392–393.
[3] S. Oramas, F. Barbieri, O. Nieto Caballero, and X. Serra, "Multimodal deep learning for music genre classification," *Transactions of the International Society for Music Information Retrieval. 2018; 1 (1): 4-21.*, 2018.
[4] C. Chen and X. Steven, "Combined transfer and active learning for high accuracy music genre classification method," in *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*. IEEE, 2021, pp. 53–56.
[5] L. K. Puppala, S. S. R. Muvva, S. R. Chinige, and P. Rajendran, "A novel music genre classification using convolutional neural network," in *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, 2021, pp. 1246–1249.
[6] E. Chaudary, S. Aziz, M. U. Khan, and P. Gretschmann, "Music genre classification using support vector machine and empirical mode decomposition," in *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*. IEEE, 2021, pp. 1–5.
[7] S. Sharma, P. Fulzele, and I. Sreedevi, "Novel hybrid model for music genre classification based on support vector machine," in *2018 IEEE symposium on computer applications & industrial electronics (ISCAIE)*. IEEE, 2018, pp. 395–400.
[8] L. Nanni, Y. M. Costa, R. L. Aguiar, C. N. Silla Jr, and S. Brahnam, "Ensemble of deep learning, visual and acoustic features for music genre classification," *Journal of New Music Research*, vol. 47, no. 4, pp. 383–397, 2018.
[9] C. Xu, N. C. Maddage, X. Shao, F. Cao, and Q. Tian, "Musical genre classification using support vector machines," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, vol. 5. IEEE, 2003, pp. V–429.
[10] E. N. Tamatjita and A. W. Mahastama, "Comparison of music genre classification using nearest centroid classifier and k-nearest neighbours," in *2016 International Conference on Information Management and Technology (ICIMTech)*. IEEE, 2016, pp. 118–123.
[11] U. Şimşekli, "Automatic music genre classification using bass lines," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 4137–4140.
[12] M. Wu and X. Liu, "A double weighted knn algorithm and its application in the music genre classification," in *2019 6th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2020, pp. 335–340.
[13] Y.-H. Cheng, P.-C. Chang, and C.-N. Kuo, "Convolutional neural networks approach for music genre classification," in *2020 International Symposium on Computer, Consumer and Control (IS3C)*, 2020, pp. 399–403.
[14] W. Zhang, W. Lei, X. Xu, and X. Xing, "Improved music genre classification with convolutional neural networks." in *Interspeech*, 2016, pp. 3304–3308.
[15] B. L. Sturm, "An analysis of the gtzan music genre dataset," in *Proceedings of the second international ACM workshop on Music information retrieval with user-centered and multimodal strategies*, 2012, pp. 7–12.
[16] B. McFee, V. Lostanlen, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, J. Mason, D. Ellis, E. Battenberg, S. Seyfarth, R. Yamamoto, K. Choi, viktorandreevichmorozov, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, D. Hereñú, F.-R. Stöter, P. Friesch, A. Weiss, M. Vollrath, and T. Kim, "librosa/librosa: 0.8.0," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3955228
[17] T. Giannakopoulos, "pyaudioanalysis: An open-source python library for audio signal analysis," *PloS one*, vol. 10, no. 12, 2015.
[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
[20] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.
[21] T. Li, M. Ogihara, and Q. Li, "A comparative study on content-based music genre classification," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003, pp. 282–289.
[22] D. S. Rahardwika, E. H. Rachmawanto, C. A. Sari, C. Irawan, D. P. Kusumaningrum, S. L. Trusthi *et al.*, "Comparison of svm, knn, and nb classifier for genre music classification based on metadata," in *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*. IEEE, 2020, pp. 12–16.
[23] D. Arora, M. Garg, and M. Gupta, "Diving deep in deep convolutional neural network," in *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2020, pp. 749–751.
[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[25] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.

APPENDIX A
CONFUSION MATRICES

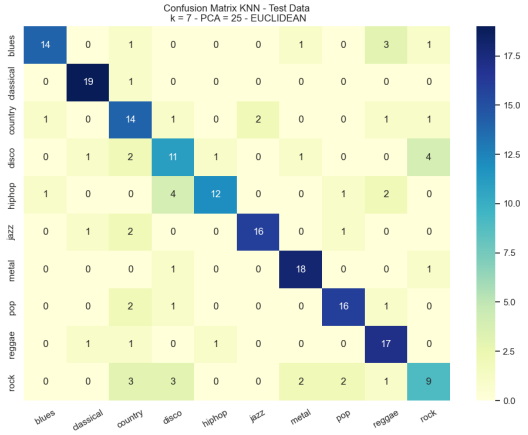


Fig. 1. Test results for KNN, k = 7 and 25 principal components

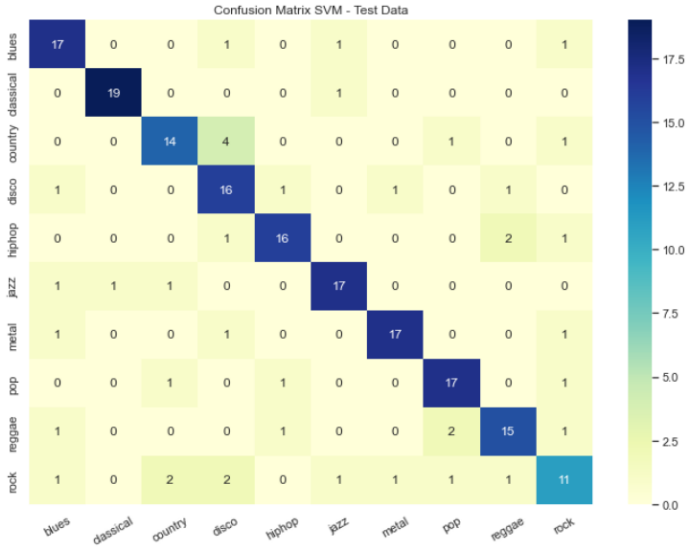


Fig. 2. Test Results for SVM

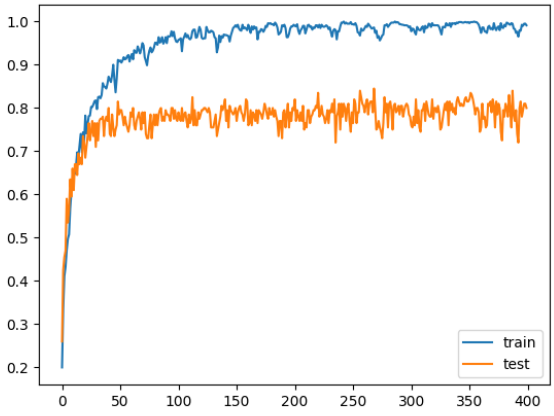


Fig. 3. Epochs vs. Accuracy - CNN

In [ ]:
```
import pandas as pd
from pandas import DataFrame
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler, QuantileTra

from sklearn.model_selection import train_test_split, cross_val_score, cros
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score, matthe
```

In [ ]:
```
train = pd.read_csv('./feature_extraction/features_last.csv')
validation = pd.read_csv('./feature_extraction/validation.csv', header=None
X = pd.read_csv('./feature_extraction/features_last.csv')
```

In [ ]:
```
train.drop(['song_name'], axis=1, inplace=True)
train.head()
```

Out[ ]:

	mean_spectral_rolloff	std_spectral_rolloff	mean_spectral_centroid	std_spectral_centroid	mean_sp
0	3805.839606	949.476395	1784.165850	360.241675	
1	3550.522098	1725.657379	1530.176679	613.066125	
2	3042.260232	885.457204	1552.811865	395.559911	
3	2184.745799	1221.963322	1070.106615	429.366909	
4	3579.757627	1254.184130	1835.004266	586.003361	

5 rows × 144 columns

In [ ]:
```
# check to see if all labels are ints
train['label'].unique()
```

Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
```

In [ ]:
```
X.drop(['song_name'], axis=1, inplace=True)
X.drop(['label'], axis=1, inplace=True)
X.head()
```

Out[ ]:

	mean_spectral_rolloff	std_spectral_rolloff	mean_spectral_centroid	std_spectral_centroid	mean_sp
0	3805.839606	949.476395	1784.165850	360.241675	
1	3550.522098	1725.657379	1530.176679	613.066125	
2	3042.260232	885.457204	1552.811865	395.559911	
3	2184.745799	1221.963322	1070.106615	429.366909	
4	3579.757627	1254.184130	1835.004266	586.003361	

5 rows × 143 columns

In [ ]:
```
y = train['label'].copy()
y.head()
```

Out[ ]:
```
0    0
1    0
2    0
3    0
4    0
Name: label, dtype: int64
```

In [ ]:
```
# Normalization of Data
scaler = StandardScaler()
X = scaler.fit_transform(np.array(X.iloc[:, :-1], dtype = float))
```

In [ ]:
```
# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, ra

# Now since we want the valid and test size to be equal (10% each of overal
# we have to define valid_size=0.5 (that is 50% of remaining data)
print(X_train.shape), print(y_train.shape)
print(X_test.shape), print(y_test.shape)
```

Out[ ]:
```
(800, 142)
(800,)
(200, 142)
(200,)
(None, None)
```

In [ ]:
```
y_train.head()
```

Out[ ]:
```
535    5
973    9
825    8
531    5
425    4
Name: label, dtype: int64
```

In [ ]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
y_test.shape
```

Out[ ]:
```
(200,)
```

In [ ]:
```
parameters = {'kernel':('linear', 'rbf', 'poly'),
              'C':[1, 10, 20]
             }

param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear', 'rbf', 'poly']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['linear',
 ]

cls=SVC()
# model.fit(X_valid, y_valid, groups=None)
model = GridSearchCV(cls, param_grid, cv=3)
model.fit(X_train,y_train,groups=None)
model
```

Out[ ]:
```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000],
                          'kernel': ['linear', 'rbf', 'poly']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['linear', 'rbf', 'poly']}])
```

In [ ]:
```
print(model.best_params_)
```

```
{'C': 100, 'kernel': 'rbf'}
```

In [ ]:
```
scoring = {'prec_macro': 'precision_macro', 'recall_score': make_scorer(reca
           'precision_score': make_scorer(precision_score, average='weighte
scores = cross_validate(cls, X, y, scoring=scoring, cv=10)
sorted(scores.keys())
scores
```

Out[ ]:
```
{'fit_time': array([0.06183004, 0.05385542, 0.05585051, 0.0538559 , 0.051861
05,
        0.06683898, 0.05505443, 0.05385876, 0.0539639 , 0.05394363]),
 'score_time': array([0.017241  , 0.01396298, 0.01496005, 0.01396275, 0.0139
6275,
        0.01696205, 0.01385641, 0.01496005, 0.01405454, 0.0139966 ]),
 'test_prec_macro': array([0.66452381, 0.67238095, 0.84692308, 0.76101265,
0.6703108 ,
        0.70440115, 0.71038295, 0.84575928, 0.6593007 , 0.81667388]),
 'test_recall_score': array([0.65, 0.69, 0.84, 0.74, 0.7 , 0.7 , 0.67, 0.82,
0.69, 0.81]),
 'test_f1_score': array([0.63838261, 0.66743597, 0.83728613, 0.73825671, 0.6
7414755,
        0.69298648, 0.6718977 , 0.82162167, 0.6525637 , 0.8093989 ]),
 'test_precision_score': array([0.66452381, 0.67238095, 0.84692308, 0.761012
65, 0.6703108 ,
        0.70440115, 0.71038295, 0.84575928, 0.6593007 , 0.81667388]),
 'test_accuracy_score': array([0.65, 0.69, 0.84, 0.74, 0.7 , 0.7 , 0.67, 0.8
2, 0.69, 0.81])}
```

In [ ]:
```
test_pred = model.predict(X_test)

np.savetxt("accuracy_solution.csv",
           np.dstack((np.arange(1, test_pred.size+1),test_pred))[0],
           delimiter=',', comments="", fmt='%i,%i',
           header="Sample_id,Sample_label")
```

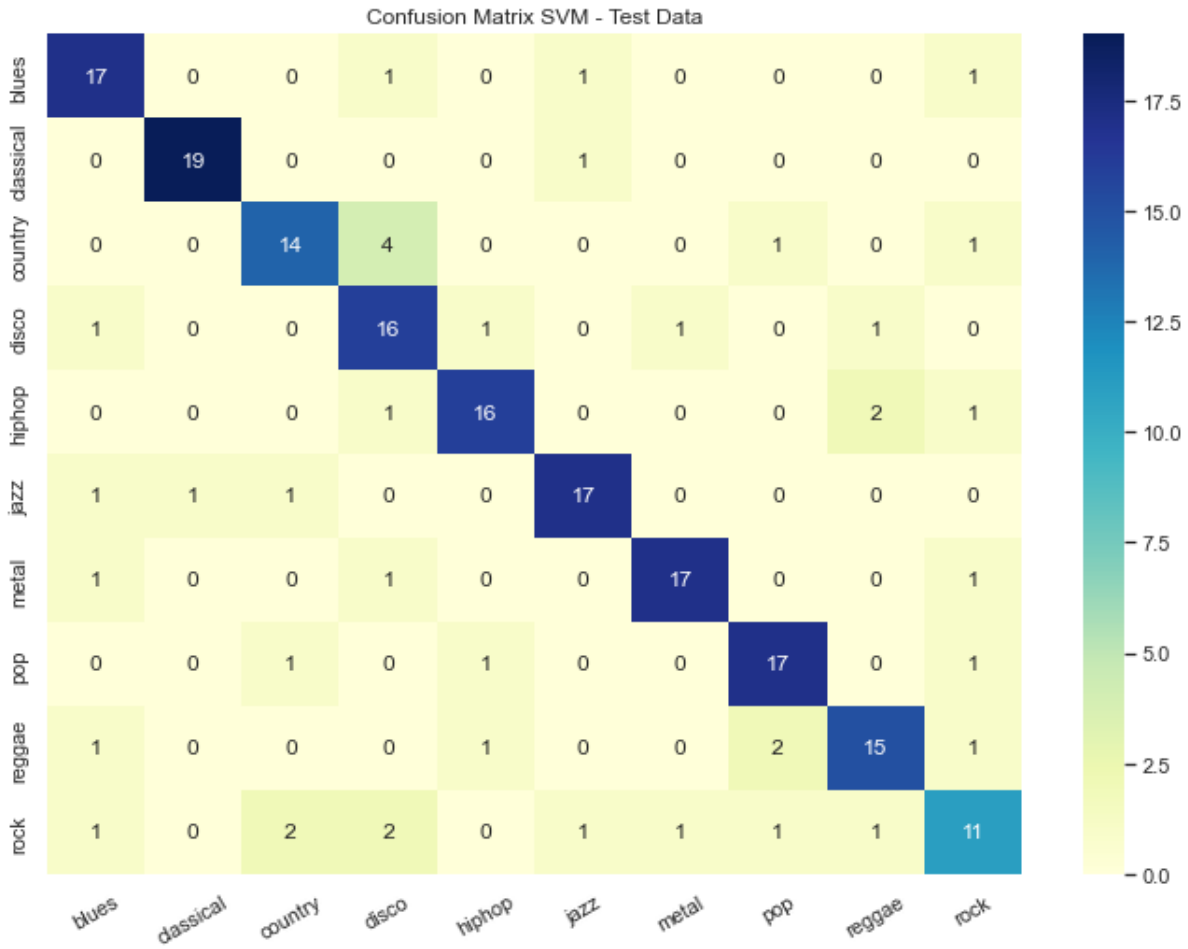
In [ ]:
```
print("Accuracy : ", accuracy_score(y_test, test_pred))
print("Balanced Accuracy : ", balanced_accuracy_score(y_test, test_pred))
print("Matthews Correlation Coefficient: ", matthews_corrcoef(y_test, test_
print("F1 Score: ", f1_score(y_test, test_pred, average = 'weighted'))
print("Precision Score: ", precision_score(y_test, test_pred, average = 'we


conf_matrix = confusion_matrix(y_test, test_pred)

df_cm = pd.DataFrame(conf_matrix, index =['blues', 'classical', 'country',
                columns = ['blues', 'classical', 'country', 'disco', 'hipho


sn.set(rc={'figure.figsize':(11.7,8.27)})
plt.title('Confusion Matrix SVM - Test Data')
ax = sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
ax.tick_params(axis='x', rotation=30)
plt.savefig('./confusion_matrix_svm_test.png')
plt.show()
plt.clf()
```

```
Accuracy :  0.795
Balanced Accuracy :  0.7949999999999999
Matthews Correlation Coefficient:  0.7727160597311248
F1 Score:  0.7942878374714062
Precision Score:  0.7973403473031955
```



<Figure size 842.4x595.44 with 0 Axes>

In [ ]:
```
print(classification_report(y_test, test_pred, digits=3))
```

	precision	recall	f1-score	support
0	0.773	0.850	0.810	20
1	0.950	0.950	0.950	20
2	0.778	0.700	0.737	20
3	0.640	0.800	0.711	20
4	0.842	0.800	0.821	20
5	0.850	0.850	0.850	20
6	0.895	0.850	0.872	20
7	0.810	0.850	0.829	20
8	0.789	0.750	0.769	20
9	0.647	0.550	0.595	20
accuracy			0.795	200
macro avg	0.797	0.795	0.794	200
weighted avg	0.797	0.795	0.794	200

In [ ]:

In [ ]:

In [ ]:

```
import librosa as lr
import numpy as np
#import pyAudioAnalysis as pya
#from pyAudioAnalysis import MidTermFeatures as mtf
from pyAudioAnalysis import ShortTermFeatures as stf
from pyAudioAnalysis import audioBasicIO as aIO

genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop'
, 'reggae', 'rock']

beeg_features = []
with open('features_sr_fixed.csv', "w+") as f:
    f.write("song_name,mean_spectral_rolloff,std_spectral_rolloff,mean_spectral_cent
roid,std_spectral_centroid,mean_spectral_bandwidth,std_spectral_bandwidth"
    + ",mean_spectral_flatness,std_spectral_flatness,mean_spectral_contrast,std_spec
tral_contrast"
    + ",mean_tempogram,std_tempogram,mean_tempo,mean_rms,std_rms,mean_zero_crossing_
rate,std_zero_crossing_rate"
    + ",mean_chroma_stft,std_chroma_stft,mean_chroma_cens,std_chroma_cens,mean_chrom
a_cqt,std_chroma_cqt"
    + ",mean_fifth_x_axis,std_fifth_x_axis,mean_fifth_y_axis,std_fifth_y_axis,mean_m
inor_x_axis,std_minor_x_axis,mean_minor_y_axis,std_minor_y_axis"
    + ",mean_major_x_axis,std_major_x_axis,mean_major_y_axis,std_major_y_axis")
    for i in range(20):
        f.write(",mean_mfcc_" + str(i))
        f.write(",std_mfcc_" + str(i))

    # dummy = '../dataset/genres/blues/blues.00000.wav'
    # [Fs, x]    = aIO.read_audio_file(dummy)
    # _, f_names = stf.feature_extraction(x, Fs, 0.050*Fs, 0.025*Fs, deltas=False)

    # for i in range(len(f_names)):
    #     f.write("," + f_names[i] + "_mean")
    #     f.write("," + f_names[i] + "_std")
    f.write('\n')

    for genre in genres:
        index = genres.index(genre)
        for i in range(100):
            wav_file = '../dataset/genres/' + genre + '/' + genre + '.' + f'{i:0>5}'
 + '.wav'
            path = './dataset/genres/' + genre + '/' + genre + '.' + f'{i:0>5}' + '.
wav'
            y, sr = lr.load(wav_file)

            # [Fs, x]    = aIO.read_audio_file(wav_file)
            # F, f_names = stf.feature_extraction(x, Fs, 0.050*Fs, 0.025*Fs, deltas=
False)

            # F = np.transpose(F)
            # G = np.mean(F, axis=0)
            # H = np.std(F, axis=0)

            # Extract Magnitude Based (timbral) features from the audio files
            # i.e. spectral rolloff, flux, centroid, spread, decrease, slope,
            # flatness, and MFCCs
            f_spectral_rolloff    = lr.feature.spectral_rolloff(y=y, sr=sr)
            f_spectral_centroid   = lr.feature.spectral_centroid(y=y, sr=sr)
            f_spectral_bandwidth  = lr.feature.spectral_bandwidth(y=y, sr=sr)
            #_,f_spectral_spread   = stf.spectral_centroid_spread(y=y, sr=sr)
            #f_spectral_decrease   = stf.spectral_entropy(y=y, sr=sr)
            #spectral_slope
            f_spectral_flatness = lr.feature.spectral_flatness(y=y)
            f_spectral_contrast = lr.feature.spectral_contrast(y=y, sr=sr)
            f_mfccs             = lr.feature.mfcc(y=y, sr=sr)

            # Extract Tempo Based Features from the audio files
            # i.e. BPM, Energy using RMS, and beat histogram
            f_tempogram         = lr.feature.tempogram(y=y, sr=sr)
            f_fourier_tempogram = lr.feature.fourier_tempogram(y=y, sr=sr)
            f_tempo             = lr.beat.tempo(y=y, sr=sr)
            f_rms               = lr.feature.rms(y=y)
```

```
            # Extract Pitch Based Features from the audio files
            # i.e. zero crossing rate
            f_zero_crossing_rate = lr.feature.zero_crossing_rate(y=y)

            # Extract Chordal Progression Features from the audio files
            # i.e. chroma
            f_chroma_stft  = lr.feature.chroma_stft(y=y, sr=sr)
            f_chroma_cens  = lr.feature.chroma_cens(y=y, sr=sr)
            f_chroma_cqt   = lr.feature.chroma_cqt(y=y, sr=sr)

            f_tonnetz      = lr.feature.tonnetz(y=y, sr=sr)


            f.write(f'{index},{np.mean(f_spectral_rolloff)},{np.std(f_spectral_rollo
ff)},{np.mean(f_spectral_centroid)},{np.std(f_spectral_centroid)},{np.mean(f_spectra
l_bandwidth)},{np.std(f_spectral_bandwidth)}')
            f.write(f',{np.mean(f_spectral_flatness)},{np.std(f_spectral_flatness)},
{np.mean(f_spectral_contrast)},{np.std(f_spectral_contrast)}')
            f.write(f',{np.mean(f_tempogram)},{np.std(f_tempogram)},{np.mean(f_tempo
)},{np.mean(f_rms)},{np.std(f_rms)},{np.mean(f_zero_crossing_rate)},{np.std(f_zero_c
rossing_rate)}')
            f.write(f',{np.mean(f_chroma_stft)},{np.std(f_chroma_stft)},{np.mean(f_c
hroma_cens)},{np.std(f_chroma_cens)},{np.mean(f_chroma_cqt)},{np.std(f_chroma_cqt)}'
)
            f.write(f',{np.mean(f_tonnetz[0])},{np.std(f_tonnetz[0])},{np.mean(f_ton
netz[1])},{np.std(f_tonnetz[1])},{np.mean(f_tonnetz[2])},{np.std(f_tonnetz[2])},{np.
mean(f_tonnetz[3])},{np.std(f_tonnetz[3])}')
            f.write(f',{np.mean(f_tonnetz[4])},{np.std(f_tonnetz[4])},{np.mean(f_ton
netz[5])},{np.std(f_tonnetz[5])}')


            for i in f_mfccs:
                f.write(',' + str(np.mean(i)))
                f.write(',' + str(np.std(i)))

            # for i in range(len(f_names)):
            #     f.write(',' + str(G[i]))
            #     f.write(',' + str(H[i]))

            f.write('\n')
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import umap
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler

sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})

df = pd.read_csv("feature_extraction/features_lr_only.csv")

X = df.drop(columns=['song_name'])
y = df['song_name'].values
#print(X.head())


scaler = MinMaxScaler()
X = scaler.fit_transform(np.array(X, dtype = float))
X = pd.DataFrame(X)

df1 = pd.DataFrame({'label': y})
data = X.join(df1)
#print(data.head())

reducer = umap.UMAP(metric='mahalanobis',
     min_dist=0.01, n_components=3)

reducer.fit(X, df1)

embedding = reducer.transform(X)
# Verify that the result of calling transform is
# idenitical to accessing the embedding_ attribute
assert(np.all(embedding == reducer.embedding_))
print(embedding.shape)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(embedding[:, 0], embedding[:, 1], embedding[:, 2], c = df1.label, cmap= '
tab10')
plt.title('UMAP projection of the GTZAN dataset\n Mahalanobis Distance', fontsize=24
)

plt.show()
```

```
import sys
import pandas as pd
import numpy as np
from scipy.spatial import distance
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.neighbors import DistanceMetric
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from scipy.spatial.distance import pdist, wminkowski, squareform
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score, matthews_corrco
ef, confusion_matrix, classification_report, make_scorer, f1_score, precision_score,
 recall_score
import seaborn as sn

used_metric     = str(sys.argv[1])
num_neighbors   = int(sys.argv[2])
pca_components  = int(sys.argv[3])
r_state         = int(sys.argv[4])
genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop'
, 'reggae', 'rock']
df = pd.read_csv('feature_extraction/features_last.csv')
X = df.drop(columns=['song_name'])  # Keeps all the features of the songs
scaler = MinMaxScaler()
X = scaler.fit_transform(np.array(X.iloc[:, :-1], dtype = float))


y = df['label'].values

if pca_components > 0:
    pca = PCA(n_components=pca_components)
    pca.fit(X)
    X = pca.transform(X)

''' Split dataset into train and test data '''
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
#X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, stra
tify=y_test)

knn = KNeighborsClassifier(metric=used_metric, n_neighbors=num_neighbors)

if used_metric == 'mahalanobis':
    knn = KNeighborsClassifier(n_neighbors=num_neighbors, metric=used_metric, metric
_params={'VI': np.cov(X_train.T)})

# this is only used for testing not reporting
if used_metric == 'wminkowski':
    distances = np.random.uniform(0, 1, X_train.shape[0])
    knn = KNeighborsClassifier(n_neighbors=num_neighbors, metric=used_metric, metric
_params={'w': distances})

''' Fit the classifier to the data '''
knn.fit(X_train, y_train)
#c_mat = confusion_matrix(y_test, knn.predict(X_test))
# print(knn.score(X_val, y_val))
# print(knn.score(X_test, y_test))


test_pred = knn.predict(X_test)

print("Accuracy : ", accuracy_score(y_test, test_pred))
print("Balanced Accuracy : ", balanced_accuracy_score(y_test, test_pred))
print("Matthews Correlation Coefficient: ", matthews_corrcoef(y_test, test_pred))
print("F1 Score: ", f1_score(y_test, test_pred, average = 'weighted'))
print("Precision Score: ", precision_score(y_test, test_pred, average = 'weighted'))
print("Recall Score: ", recall_score(y_test, test_pred, average = 'weighted'))

conf_matrix = confusion_matrix(y_test, test_pred)
```

```
df_cm = pd.DataFrame(conf_matrix, index =['blues', 'classical', 'country', 'disco',
'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock'],
                columns = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz
', 'metal', 'pop', 'reggae', 'rock'])


sn.set(rc={'figure.figsize':(11.7,8.27)})
plt.title('Confusion Matrix KNN - Test Data\n k = ' + str(num_neighbors) + ' - PCA =
 ' + str(pca_components) + ' - ' + used_metric.upper())
ax = sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
ax.tick_params(axis='x', rotation=30)
plt.savefig('./confusion_matrix_knn_test.png')
#plt.show()
plt.clf()


# burcu 15
# esad 10
# emin 25
# aleyna 20
```

```
# inspired from https://github.com/Shashabl0/Music-Genre-Classification-using-CNN/blob/master/music-genre-classification.ipynb

# lets import libraries that we will be usings
# we already have imported pandas and numpy
import sys
from keras.backend import dropout
import keras as k
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


epohcs   = int(sys.argv[1])
batch_size  = int(sys.argv[2])
dropout = float(sys.argv[3])

# Loading dataset
# we have 2 CSVs here, one containing features for 30 sec audio file, mean & variance for diff features we have, then
# and one for 3 sec audio files. I will be using 3 sec audio
dataf = pd.read_csv('feature_extraction/features_lr_only.csv', skiprows=1, header=None)

y = dataf.iloc[:,0]

# scaling features
from sklearn.preprocessing import MinMaxScaler
fit = MinMaxScaler()
X = fit.fit_transform(np.array(dataf.iloc[:,1:],dtype=float))

# dividing into training and test Data
X_train,x_test, Y_train, y_test = train_test_split(X,y,test_size=0.2,shuffle=True)


# Using CNN algorithm
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint

# building model
model = k.models.Sequential([
    k.layers.Dense(512,input_shape=(X_train.shape[1],),activation='relu'),
    k.layers.Dropout(dropout),

    k.layers.Dense(256, activation='relu'),
    k.layers.Dropout(dropout),

    k.layers.Dense(128, activation='relu'),
    k.layers.Dropout(dropout),

    k.layers.Dense(64, activation='relu'),
    k.layers.Dropout(dropout),

    k.layers.Dense(10, activation='softmax'),
])


model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics='accuracy')

#earlystop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=10,min_delta=0.0001)
#modelcheck = ModelCheckpoint('best_model.hdf5',monitor='val_accuracy',verbose=1,save_best_only=True,mode='max')

model.fit(X_train,Y_train, validation_data=(x_test,y_test), epochs=epohcs, batch_size=batch_size,verbose = 0)


# from matplotlib import pyplot
# pyplot.plot(history.history['loss'], label='train')
```

```
# pyplot.plot(history.history['val_loss'], label='test')
# pyplot.legend()
# pyplot.show()

test_loss, test_accuracy = model.evaluate(x_test,y_test,batch_size=batch_size)
print("Test loss : ",test_loss)
print("Best test accuracy : ",test_accuracy*100)
print("\n\n")
```

```
echo "Metrics,Num_Neighbors,PCA_Components,N_Random_States,Score" > log/logs8.csv

metrics="minkowski mahalanobis euclidean manhattan"

for metric in $metrics
do
    for neighbor in 1 2 3 4 5 6 7 8 9 10 11 13 15 20
    do
        for PCA_Components in 0 3 4 5 6 7 8 9 10 13 15 20 25 30 35 40 45 50 55 60 65
 70 75
        do
            for state in 1 2 3 5 10 15 20
            do
                echo -n "${metric},${neighbor},${PCA_Components},${state}," >> log/l
ogs8.csv
                python knn_impl.py $metric $neighbor $PCA_Components $state >> log/l
ogs8.csv
            done
        done
    done
done
```

```
echo "epochs,batch_size" >> log/logs5.csv

for epochs in 100 200 400
do
    for batch_size in 32 64 128 512
    do
        for dropout in 0.1 0.2 0.3
        do
            echo -n "${epochs},${batch_size},${dropout}" >> log/logs5.csv
            python cnn.py $epochs $batch_size $dropout  >> log/logs5.csv
        done
    done
done
```