# AUTOMATING THE IDENTIFICATION OF SIMILAR SONGS USING AUTOENCODERS AND COMMUNITY DETECTION

*Aisha Dantuluri, Tyler Farnan and Soumyaraj Bose*

University of California San Diego, La Jolla, CA 92093-0238, USA

## ABSTRACT

We propose an unsupervised approach to music classification through the use of clustering mel-spectrograms of audio clips. Optimally the clusters will classify songs in communities of audio files having similar musical structures. To evaluate the performance of the classification we then compare clusters with graph communities of the same songs based on the qualitative features associated with them, generated from playlist metadata for playlists that the songs appear on.

***Index Terms*—** Autoencoder, Convolutional Neural Network, Unsupervised, Clustering, Community Detection

## 1. INTRODUCTION

This project delivers an insight into how a potential music recommendation engine (MRE) analyzes a large dataset of songs and categorizes them, as per their attributes, with as little information about them as possible. It leverages the "listening ear" of an MRE to observe the frequency-time distribution of songs and cluster sets of them based on machine generated representations, without using any tags or metadata (i.e. an unsupervised approach). The feature extraction is the aspect that distinguishes this work which attempts the unsupervised classification of audio samples. It varies from using Hidden Markov models [1] to analyzing statistical spectrum descriptors (SSDs) of the audio's attributes [2].

### 1.1. Related Work

Although similar to works such as [2], we derive primary inspiration from DEEJ-A.I.[3]. The framework and application of [3] is that of a traditional MRE as it extracts features from song samples available from the Spotify Web API and, recommends songs based on their learnt attributes. It uses convolutional autoencoders to learn and produce the latent representations of their spectrograms.

[3] uses cosine proximity as the basis of grouping songs with similar latent forms. This project chooses a combination of unsupervised approaches, akin to [2] and [4], to allow the machine to cluster similar songs without being affected by any external metrics. This work also deviates from [3] by using cluster analysis to get the optimum number of clusters.

### 1.2. Dataset

Two different datasets are utilized for this study. The first is a collection of Spotify playlists and the ID of each song contained in that playlist (playlist dataset). Each song appears in a minimum of 10 playlists. The second component is a collection of 5 second audio samples, represented as mel-spectrograms, (audio dataset). Both are sourced from [3].

#### *1.2.1. Preprocessing*

The playlist dataset is used to generate a network of songs using the functionality of the `networkx` [5] module. This network connects each song to those sharing the same playlists in this dataset. The largest connected subgraph of this network serves to provide the IDs of those songs whose spectrograms are then used for training the architectures concerned.

The mel-spectrograms of song samples from the project dataset are stored as `.png` image files. These images are converted to 3D `numpy` arrays in the format [batch-size, width, height] and scaled by their minimum and maximum values using the `MinMaxScaler` class of `scikit-learn` [6]. The scaled training and validation sets are converted to 4D arrays in the format [batch-size, width, height, channels] and, then flattened to 2D to set up for faster, memory-efficient training.

#### *1.2.2. Features*

We extract features from the spectrogram data using an autoencoder. This is a feature extraction technique used widely in image processing but, in recent years, has also been employed for audio analysis [7] and [8]. We use an autoencoder to replicate the spectrograms. As they are large arrays with multiple redundancies, we can generate compact representations of the data with the redundancies eliminated.

Further, using the playlist metadata, we will construct communities which represent similar songs, which will serve as ground truth for comparison with our generated clusters.

## 2. METHODS

### 2.1. K-means Clustering

After collecting the encoder's latent vector representations of the spectrograms in the validation set, we perform the **K-means clustering algorithm** [9] where K are the number of partitions, $x_i$ are the data points and $m_k$ are the centroids.

$$\sum_{k=1}^{K} \sum_{i \in c_k} ||x_i - m_k||^2 \qquad \text{Euclidean distance}$$

### 2.2. Optimal Cluster Analysis

#### *2.2.1. Silhouette Scoring*

The silhouette [10] of a data point quantifies its similarity to other points in its determined cluster in comparison to other clusters in the data space. The score can take values in $[-1, 1]$, where $+1$ and $-1$ indicate perfect and very poor match to a cluster, respectively. The silhouette score is the maximum of the mean silhouette value for the entire dataset for $k$ clusters.

#### *2.2.2. Elbow Method*

The Elbow method [11] is a heuristic to determine in the optimal number of clusters in a given dataset. Upon clustering, a certain set of properties explain the relation of data point with respect to cluster centers in the data space (e.g. distortion, inertia), that are together termed as expected variation. Upon plotting the expected variation against the number of clusters $k$, the Elbow method dictates that the optimal number of clusters corresponds to the value of $k$ at which the slope of the plot transitions from a high to a low value (the elbow of the plot).

### 2.3. Community Detection

In this project we leveraged the leading eigenvector community detection method contained within the python-igraph module [12]. Given a desired number of communities, this recursive algorithm splits the graph by maximizing the modularity with respect to the input graph. Please refer to Newman's paper [12] for further details on the method.

### 2.4. Dimensionality Reduction

Data Visualization is very helpful for conducting comparative analysis between K-means labels and Community detection labels of the distributions of latent vectors. We leveraged both t-SNE, Stochastic Neighborhood Embedding [13] as well as PCA, Principal Component Analysis [14]. t-SNE is a probabilistic method for finding a low-dimensional mapping, whereas PCA is purely a mathematical method.

#### *2.4.1. t-SNE*

t-SNE minimizes the Kullback-Leibler divergence between the joint probabilities $p_{ij}$ in the high dimensional space and the joint probabilities $q_{ij}$ in the low dimensional space. [13]

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{1}$$

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}} \tag{2}$$

$$\begin{aligned} KL(P\|Q) &= \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \\ &= \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij} \end{aligned} \tag{3}$$

#### *2.4.2. Principal Component Analysis*

Principal Component Analysis (PCA) [14] is a dimensionality-reduction technique: on application to a multivariate dataset, it reduces the number of variables without losing most of their information. This is achieved by iteratively generating a linear best fit of the data, thus producing an orthogonal basis of dimensions that are uncorrelated (the *principal components*).

The covariance matrix $\sum$ of the standardized original dataset $X$ is computed which is, then, subjected to an eigenvalue decomposition of the form

$$A = Q \sum Q^{-1} \tag{4}$$

Equation (4) returns eigenvectors $v$ corresponding to a set of eigenvalues $\lambda$ ($dim(\lambda) \leq N$ or the original number of dimensions). The $v$ are sequenced in decreasing order of their corresponding $\lambda$. The algorithm then gets rid of the $v$ of least emphasis and creates fills the columns of the feature vector $F$ with the remaining. Ultimately, the final dataset is generated from projecting $X$ onto the feature vector space

$$X_{final} = X * F \tag{5}$$

## 3. MODELS

### 3.1. Autoencoder

The spectrogram data includes points of size $(96 \times 216)$. The use of convolutional layers will avoid overfitting the models and allow for the creation of deep enough models to generate compact representations. To perform dimensionality reduction on the data, we constrain the AE to generate compact representations, smaller than the above size. We choose the number of layers and layer sizes accordingly.

| | Model A0 | Model A1 | Model A2 | Model A3 |
|---|---|---|---|---|
| L1 | [48, 108, 96] | [48, 108, 48] | [48, 108, 96] | [48, 108, 48] |
| L2 | [24, 54, 45] | [24, 54, 24] | - | [24, 54, 24] |
| L3 | - | - | - | [12, 27, 12] |
| LV | [12, 27, 16] | [12, 27, 12] | [24, 54, 14] | [4, 9, 5] |
| LV' | [72, 72] | [48, 81] | [108, 168] | [12, 15] |

**Table 1**. The intermediate output layers for all encoder models tested indicates the number of filters applied at every level of the model (by observing the 3rd dimension) and these differences when combined with differing numbers of layers generate different sized feature vectors. L1, L2 and L3 indicate the layer module outputs of the encoder, LV and LV' are the latent vector and the same vector reshaped.

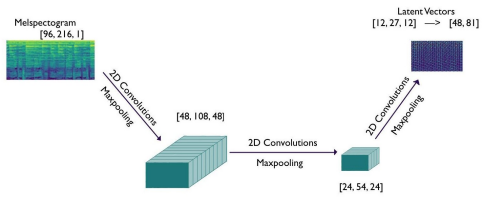


**Fig. 1**. Illustration of Model A1 and its latent vector construction from a spectrogram

The primary model we employ is Model A0. The encoder is built with 3 separate layer modules, each module consisting of a 2D convolutional layer, a ReLU activation function, and a maxpooling layer. For the layers, number of filters used were 96, 48, and 16 respectively. The decoder is built in the similar manner with the order of the number of filters reversed and and with upsampling layers replacing the maxpooling layers. All convolutions are performed with $3 \times 3$ sized kernel.

The output of the encoder is a latent vector with dimensions $12 \times 27$, reshaped to size $72 \times 72$.

#### *3.1.1. Model Selection*

To pick the best model, variations on Model A0 were tested, labeled models A1, A2, and A3. For each of the models have a different number of layer modules and filter sizes, and build the optimal model by rigorously tuning the hyperparameters. This will be further elaborated in Section 4.1. 1 indicates the choice of model specific architecture. Models A1, A2 and A3 have 3, 4 aand 2 layer modules in their respective encoders.Model A0 is more complex than A1, A2 and A3. These latter three were selected to be equally complex, so as to determine the size latent vector that generates the best result. 1 shows the basic architechture of Model A1.

The performance of these models in clustering will be discussed in upcoming sections. All models are built in Python using functionality from Keras and Tensorflow.

### 3.2. Variational Autoencoder

Model B derives from the architecture of Model A0, but includes two new layers, one each in the output of the encoder and decoder networks and unique to variational autoencoders (VAE) [15]. VAEs are closest to a human-like approach to learning as they use recollection to recreate the input.

To supplement the recollection, the encoder output of the VAE must be a layer that generates a (prior) distribution corresponding to each feature learned by the encoder. Features of this latent layer distribution are then learnt by the decoder. The learnt distribution serves as the basis for the generation of samples that ultimately represent the decoded output.

The encoder of Model B adds a MultivariateNormal distribution layer sized according to the number of dimensions in the latent space ( = 16). The MultivariateNormal is corrected against a prior normal distribution using their KL divergence.

The decoder uses 2D Transpose convolutional layers with upsampling to learn the latent distributions. The mean and variance of these distributions are used to parameterize an Independent Bernoulli distribution to generate the decoding. The reconstruction is corrected using negative log-likelihood.

The VAE framework used functionality from Tensorflow, Keras and Tensorflow Probability [16]. Some design choices for Model B were directly derived from the Tensorflow Probability example for a VAE trained on MNIST [17].

## 4. EXPERIMENTS

### 4.1. Hyperparameter tuning

For each model we tested on a dataset consisting of $10,800$ points. Models A1 and A2 were trained and evaluated extensively with combinations of learning rate $\eta = \{0.001, 0.0005\}$, batch-size $b = \{5, 10, 15\}$ and activation functions ReLU and sigmoid. After obtaining similar results for these two models finally all models were trained for between 80 and 125 epochs with the Adam optimizer and MSE loss with choice of $\eta = 0.0005$, $b = 5$ and activation function ReLU.

### 4.2. Comparative Cluster analysis

For each model in our analysis, we implemented the following procedure for cluster analysis: **First**, we used techniques such as silhouette scoring to approximate the optimal number of number of clusters (K) in K-means. **Second**, we ran community detection on the playlist metadata network, with number of communities equal to the K number of clusters. **Third**, after running K-means on the latent vector spectrogram representations output by the encoder. **Fourth**, we use both T-SNE and Principle Component Analysis to visualize the the K clusters of the latent vectors. During visualization we use two kinds of labels for the songs represented: K-means cluster and community detection. **Fifth**, we use a confusion matrix as our metrix to better understand how songs grouped by

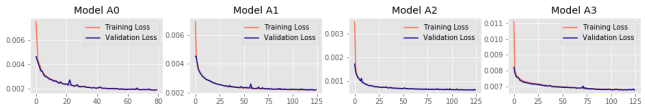


**Fig. 2**. Training and Validation loss for all four CAE models. All models were trained on optimal hyperparameters chosen from fine-tuning

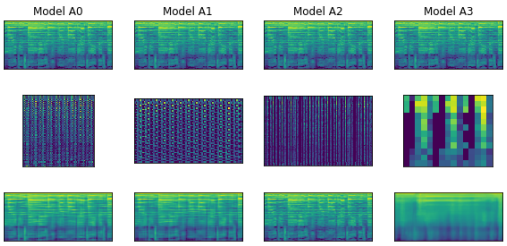


**Fig. 3**. Representations for CAE

K-means on encoded spectrograms compare with the same songs grouped by community detection on playlist meta-data.

## 5. RESULTS

### 5.1. Model Performance

All models minimize loss effectively as per Figure 2. While one of our early questions was whether minimization of validation loss would be a metric for how clustering analysis performs, we remain unable to remark conclusively upon that question. Since we see very little difference in the loss minimization trajectories, we were not able to determine marked differences between comparative clustering for the models A.

#### *5.1.1. Latent Vectors*

However, as Figure 3 indicates the models do not all perform equally when reconstructing the spectrograms (Figure 4 displays the result of recreation from recollection which is often a tricky endeavour), implying that there is a greater loss of information in the deeper autoencoders. Yet this too fails to create differences in the results of our analysis.

### 5.2. Discussion

Overall, there is very poor separability between the clusters of latent vectors learned by K-means as observed in figures.

**Fig. 4**. VAE: Input (L), Latent State (M), Output (R)

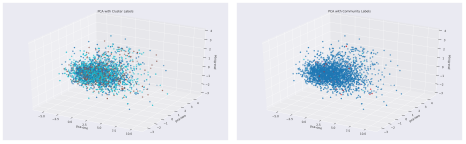

**Fig. 5**. Visualizing principal components of latent vectors labelled by k-means clustering of spectrogram representations (L) vs. detection of (4) communities of playlist metadata (R)

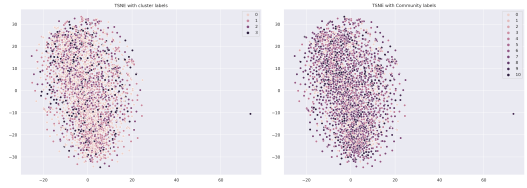

**Fig. 6**. Visualizing t-SNE manifold of latent vectors labelled by k-means clustering of spectrogram representations (L) vs. detection of (10) communities in the playlist metadata (R)

5 and 6. This lack of clustering may be due to a combination scaling the data before training and reshaping vectors before clustering. Additionally, temporal characteristics of music are lost when using purely convolution layers. It is likely that the lack of supervised signal significantly hinders an autoencoder's ability to learn "classes" of latent vectors. When comparing both confusion matrices in 7 we see that a larger number of communities is misleading. Running community detection on the playlist metadata with only 4 communities results in a highly unbalanced distribution across songs, with no correlation between the poorly separated K-means clusters and the highly unbalanced communities.

## 6. CONCLUSION

We propose an unsupervised learning approach to music classification that analyses both convolutional and variational autoencoders. We employ K-means for clustering the learned latent representations of audio features produced by the encoder. Using dimensionality reduction, we evaluate the correlations between these clusters with communities detected in a network of playlist metadata. Future work may include constructing a supervised learning approach that leverages community detection to generate labels for mel-spectrograms.

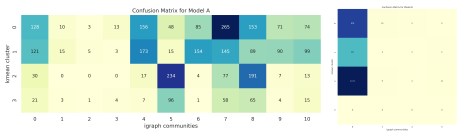

**Fig. 7**. Comparing the K-means clusters to 10 communities (left) versus 4 communities (right)

## 7. REFERENCES


[1] X. Shao, C. Xu, and M.S. Kankanhalli. Unsupervised classification of music genre using hidden markov model. *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, 3:2023–2026, 4.

[2] L. Barreira, S. Cavaco, and J.F. da Silva. Unsupervised music genre classification with a model-based approach. In *EPIA 2011: Progress in Artificial Intelligence*, pages 268–281, 2011.

[3] Robert D. Smith. Create automatic playlists by using deep learning to listen to the music. *Towards Data Science*, 2019.

[4] W. Tsai and D. Bao. Clustering music recordings based on genres. In *2010 International Conference on Information Science and Applications*, pages 1–5, 2010.

[5] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011.

[7] Jingyi Shen, Runqi Wang, and Han-Wei Shen. Visual exploration of latent space for traditional chinese music. *Visual Informatics*, 4, 04 2020.

[8] Colonel Joseph, Christopher Curro, and Sam Keene. Autoencoding Neural Networks as Musical Audio Synthesizers. *arXiv e-prints*, page arXiv:2004.13172, April 2020.

[9] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[10] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[11] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18:267–276, 1953.

[12] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[13] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[14] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[15] Carl Doersch. Tutorial on variational autoencoders, 2016.

[16] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. Tensorflow distributions, 2017.

[17] Joshua V. Dillon Ian Fischer, Alex Alemi and the TFP Team. Variational autoencoders with tensorflow probability layers, 2019.


## 8. INDIVIDUAL CONTRIBUTIONS

**T**yler Farnan helped to set up the model training pipeline and helped to implement community detection. He designed and implemented the comparative cluster analysis procedures.

**Aisha Dantuluri** used model training pipeline to set up and run experiments and tune hyperparameters to generate optimal models and ran cluster analysis procedures on the generated models.

**Soumyaraj Bose** used model training pipeline to set up and run experiments on a Variational Autoencoder model, with constant consultation on design from Tyler and Aisha, tuned hyperparameters to generate the optimal model and ran cluster analysis procedures on the generated models. He also did the preprocessing on the playlist dataset and generated the song network, with existing code from Tyler.

## 9. REPLIES TO CRITICAL REVIEWS

### 9.1. Group 81 Reviews

**Review:** Were any other clustering techniques explored, such as K-mediods or Osort clustering? The distance metric used by K-means could lead to different results as well, and perhaps commenting on the performance variations with euclidean, manhattan, mahalanobis, etc, would add value to the work.

**Response:** One idea for future work could be to use more graph analysis algorithms to find song's with a high degree in the network (for each community). Then, these corresponding latent vectors could be used as the medoids in K-medoids.

**Review:** I understand that the processing of audio features was performed by using melspectograms, but is there any way that you can use temporally-sensitive network models (such as recurrent neural networks) to learn latent features

of the data? Because music has a natural time component to it, perhaps some commentary on using recurrent autoencoders could show interesting results.

**Response:** We considered using a hybrid recurrent-convolutional architecture to capture temporal properties in the music data. However, this type of model is most typical used for supervised problems. We chose to focus on an Unsupervised problem, and opted to train a more simple model.

**Review:** Some brief explanation on what TSNE is would be nice to help the viewer understand a bit more clearly the results shown on Slide 12.

**Response:** TSNE is a probabalistic approach to visualizing high-dimensional data. The algorithm attempts to minimize the KL divergence between the joint probabilities of the low-dimensional embeddings and the original data. We've provided a description in the method sections, along with a reference to the original paper.

### 9.2. Group 82 Reviews

**Review:** It would be a good idea to include some plots of training vs validation error, some clustering results after different numbers of epochs, etc to show how your model improved as you trained it, or how it didn't and an attempt at an explanation for why it behaved the way it did.

**Response:** While these results were not formally included in our analysis, it was our observation that once the loss initially converged to the minimum, training for additional numbers of epochs did not improve the performance of the model. We discuss our theory on the loss of temporal features of the data in the Discussion section and how this may have impacted our result. Training and validation loss are included above in Figure 2

**Review:**If you could draw any connection (or lack thereof) between the success of the autoencoder and the failure of the clustering, it would be very interesting to hear why such a discrepancy might arise.

**Response:**The failure of the clustering seems not to be correlated with the success of the autoencoder in recreating the spectrograms or minimizing the loss. This is indicated by the marked difference in the reconstructions indicated in Figure 3. Rather, the features learned by the Convolutional AE do not accurately represent the temporal structure of the data. Further strictly the audio data with the lack of a supervised signal may be too little data to accurately cluster the data given that we are essentially attempting an unsupervised approach to a classification of sorts.

### 9.3. Group 19 Reviews:

**Review:** Why choose such kind of loss functions?

**Response:** For reconstruction we are trying to recreate the exact real valued pixel values of the original spectrogram or something with pixel values as close to the original as possible. In this case it makes sense to use a loss function that is a real valued output, distance metric such as MAE or MSE.

**Review:** What was the reason for choosing this CAE model architecture?

Convolutional layers were chosen to build models with fewer model weights and to avoid overfitting the models as we built deep models to generate compact representations. Further convolutional layers are excellent at extracting features from the data.

**Review:** Also, maybe given some example of the results of the cluster is better (for example, song A and song B is clustered into together

**Response:** As the generated clusters are so large and so tangled together, there is no way to determine if specific songs are always clustered together.