Music genre classification with neural networks

Pinak Mandal (pinak.mandal@icts.res.in)

November 2017

Introduction

Music genre classification is a problem in the field of music information retrieval which has found many popular applications in modern times. Fluidity of modern music and the subjective nature of genres make it a difficult task.

Data set

This work aims to analyze the GTZAN Genre Collection (http://marsyasweb.appspot.com/download/data_sets/) which was used in the well-known paper "Musical genre classification of audio signals" by G. Tzanetakis and P. Cook (2002) which has since become a standard dataset in music analysis. The dataset consists of 10 genres blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock with 100 tracks of length 30 seconds per genre in .au format.

Workflow

1. First we convert the tracks to .wav format for easier manipulation (implemented in convert.py).
2. We extract MFCC features for each track and tabulate them in csv files (implemented in process.py).
3. We do principal component analysis on the processed data and try to infer some characteristics of individual genres.
4. We divide the data into training and test sets (implemented in process.py).
4. We code a feedforward artificial neural network, feed it the training data and create a model for genre prediction (implemented in main.py).
5. We test our test data with the computed model and calculate accuracy of the model (implemented in main.py and test.py).

Code for this project is available at the github repository pinakm9/musis (link: https://github.com/pinakm9/musis).

Feature extraction

Mel-frequency cepstrum coefficients or MFCCs are commonly used as features in speech recognition systems, such as the systems which can automatically recognize numbers spoken into a telephone. The European Telecommunications Standards Institute in the early 2000s defined a standardized MFCC algorithm to be used in mobile phones[1]. Below we describe the essence of MFCC.

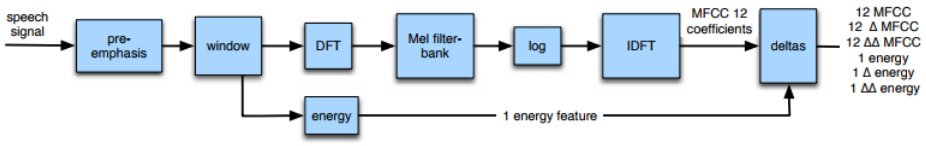


Figure 1: Extracting a sequence of 39-dimensional MFCC feature vector from audio signal

Step 1

The first processing step is the computation of the frequency domain representation of the input signal. This is achieved by computing the Discrete Fourier Transform. The audio signal is first windowed (usually Hamming window is used to avoid discontinuities) and then we apply FFT on it.

Step 2

The results of the FFT will be information about the amount of energy at each frequency band. Human hearing, however, is not equally sensitive at all frequency bands. It is less sensitive at higher frequencies, roughly above 1000 Hertz. It turns out that modeling this property of human hearing during feature extraction improves speech recognition performance. A mel is a unit of pitch defined so that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels. The mapping between frequency in Hertz and the mel scale is linear below 1000 Hz and the logarithmic above 1000 Hz. The mel frequency can be computed from the raw acoustic frequency as follows:

$$\mathrm{mel}(f) = 1127 \ln\left(1 + \frac{f}{700}\right)$$

During MFCC computation this intuition is implemented by creating a bank of filters which collect energy from each frequency band with 10 filters space linearly below 1000 Hz and the remaining filters spread logarithmically above 1000 Hz. Finally we take the log of each of mel spectrum values. In general humans are less sensitive to slight differences in amplitude at high amplitudes than at low amplitudes. In addition, using log makes the feature estimates

less sensitive to variations in input (for example power variations due to the speaker's mouth moving closer or further from the microphone).

Step 3

Cepstrum is formally defined as the inverse DFT of the log magnitude of the DFT of a signal. Instead of the IDFT the inverse discrete cosine transform is applied to the output of step 2 during MFCC computation. For purposes of MFCC extraction, we generally just take the first 12 cepstral values.

Step 4

We next add a thirteenth feature: the energy. The energy in a frame is the sum over time of the power of the samples in the frame, thus for a signal x in a window from time sample t_1 to time sample t_2, the energy is

$$\sum_{t=t_1}^{t_2} x^2[t]$$

Another important fact about the speech signal is that it is not constant from frame to frame. This change, such as the slope of a formant at its transitions, or the nature of the change from a stop closure to stop burst, can provide a useful cue for phone identity. For this reason we also add features related to the change in cepstral features over time. We do this by adding for each of the 13 features (12 cepstral features plus energy) a delta or velocity feature, and a double delta or acceleration feature. A simple way to compute deltas would be just to compute the difference between frames; thus the delta value for a particular cepstral value $c(t)$ at time t can be estimated as:

$$\Delta c(t) = \frac{c(t+1) - c(t-1)}{2}$$

and one more application of Δ on $\Delta c(t)$ gives us double delta values or $\Delta^2 c(t)$ for $c(t)$. After adding energy, delta and double delta features we end up with $(12 + 1) \times 3 = 39$ MFCC features.

Implementation

We use python_speech_features library by user jameslyons available on github (https://github.com/jameslyons/python_speech_features) to compute MFCC during our experiments.

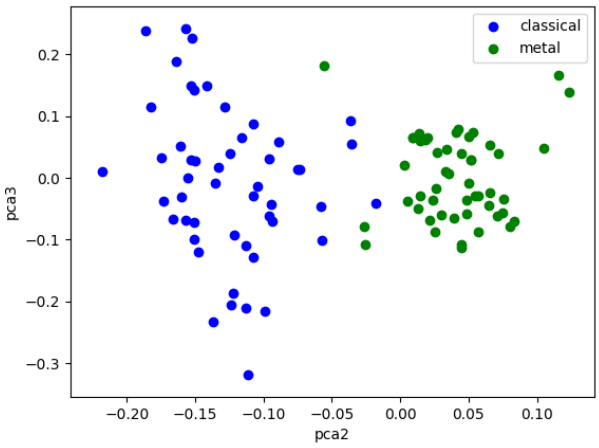
Dimension reduction

Audio files in the GTZAN database are in .au format. So we use the SoX utility to convert these files in .wav format for easier manipulation with Python libraries. Each audio snippet in the database could be represented as 30 seconds

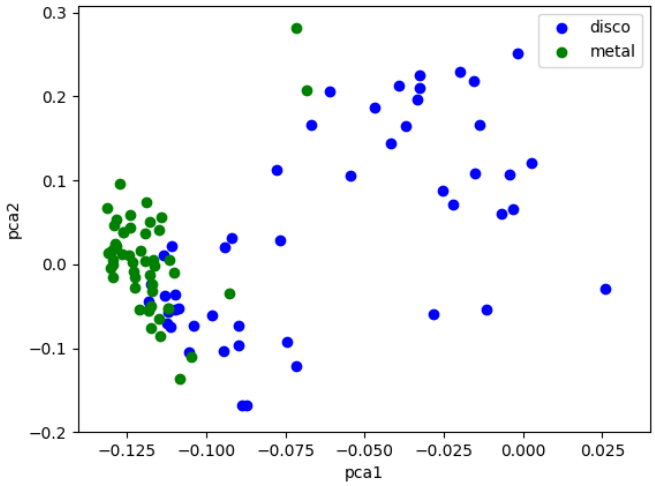
$\times$ 22050 samples/second = 661500 length vector, which would be heavy load for a convention machine learning method. We ignore the delta and double delta values during MFCC computation and end up with a $\approx 3000 \times 13$ length feature vetor for each track after computing MFCC. We further divided the MFCC features into 4 roughly equal sized sections and extracted first 40 of each section. So we ended up with a $13 \times 160 = 2080$ length of MFCC feature vector to represent a 30 second audio file for our experiments.

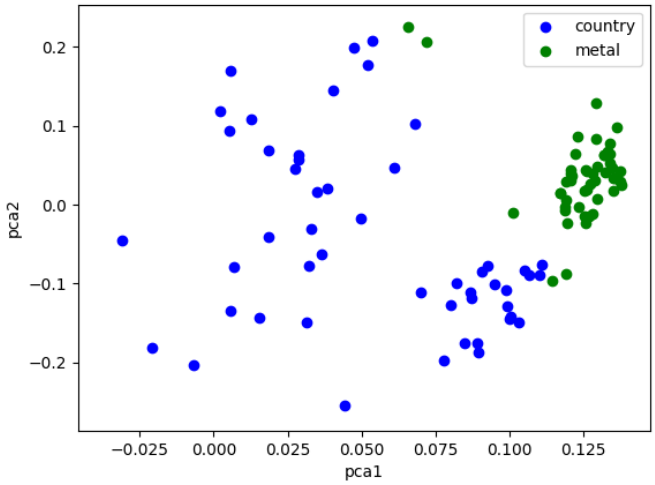
PCA on final features

To test if MFCCs have classification capability for various music genres we randomly select 500 tracks with equal weight per genre from the database, compute 2080 length MFCC vector for each of them and then compute PCA for this 500×2080 dimensional data. Not every genre is equally esay/hard to identify. Among our genres classical and metal seem to be far apart judging by distinguishability and it shows in the PCA plots.

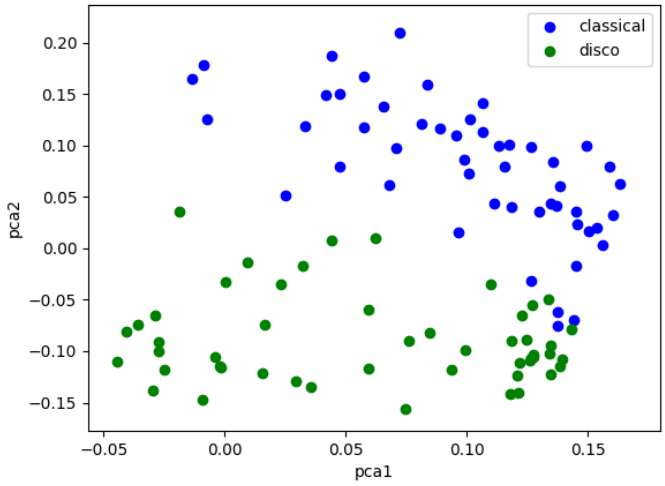


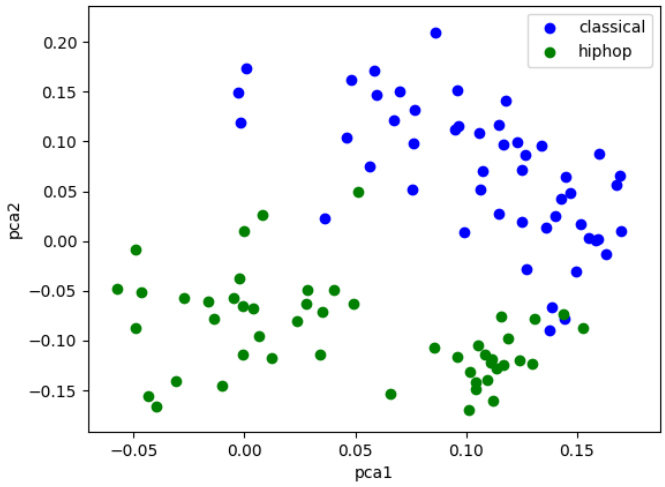
In fact metal seems to be one of the more distinguishable genres and separates nicely from other genres in PCA plots.



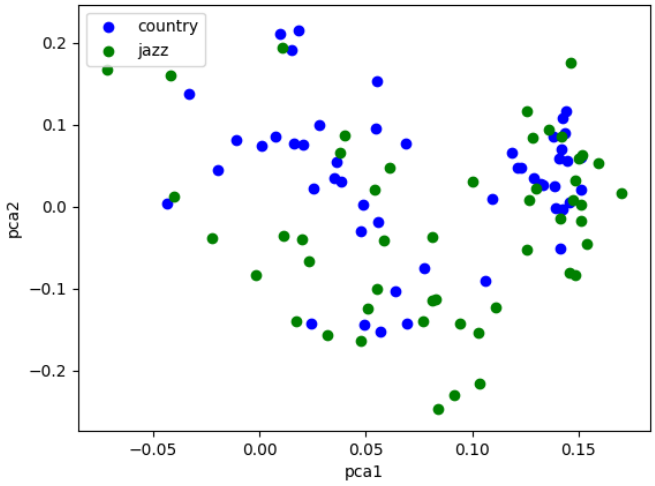


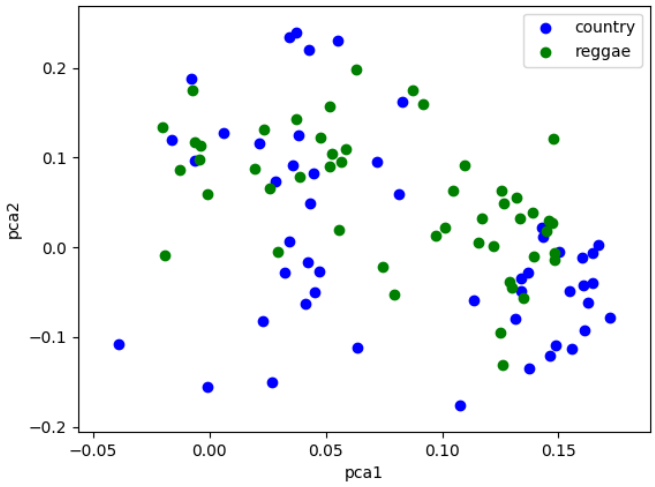
Classical is another genre with high distinguishability.



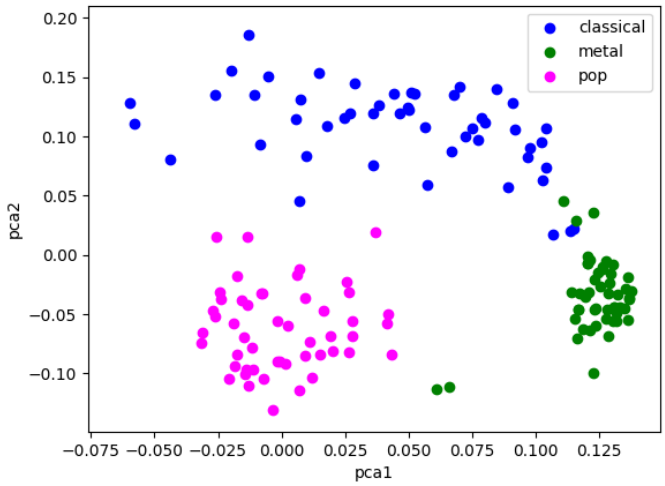


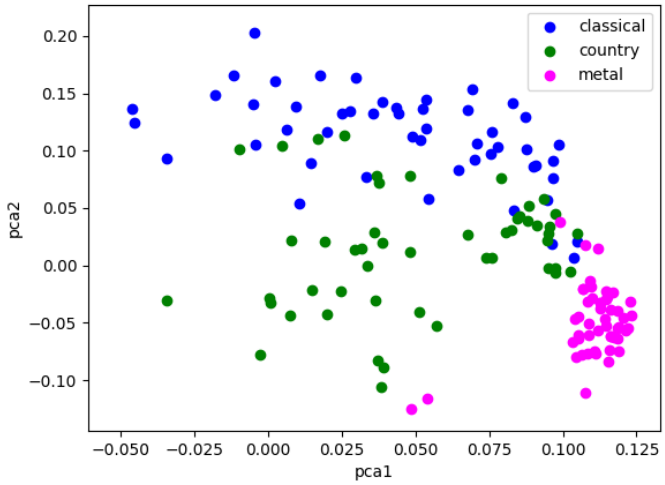
Country on the other hand has low distinguishability.



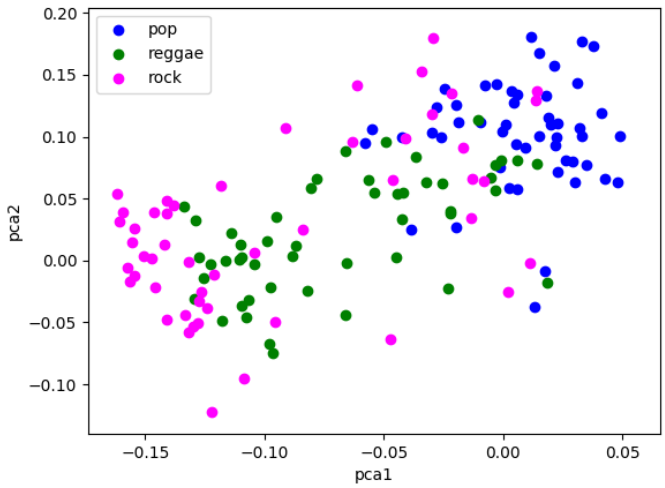


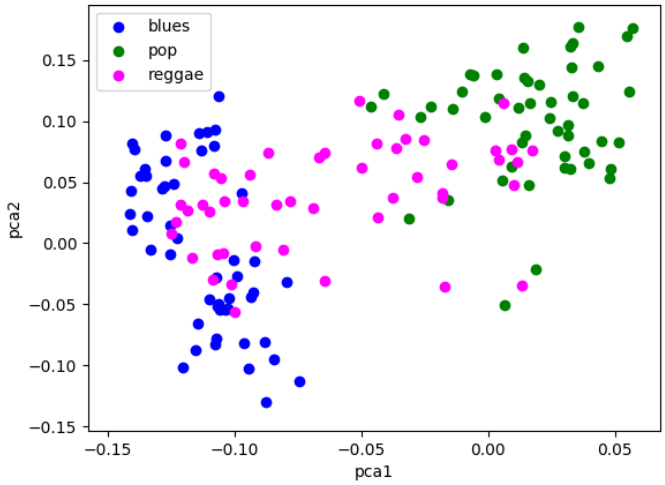
Most 2-genre groups separate in PCA plots. 3-genre groups also display some separability under PCA. Inclusion of more distinguishable genres increase separability as is seen in the plots below.



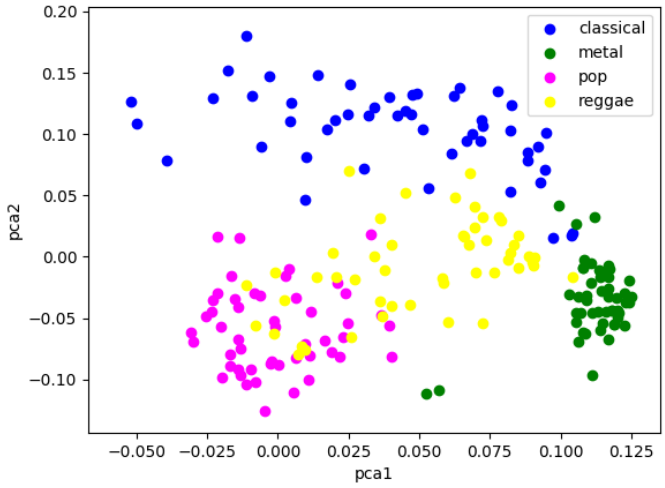


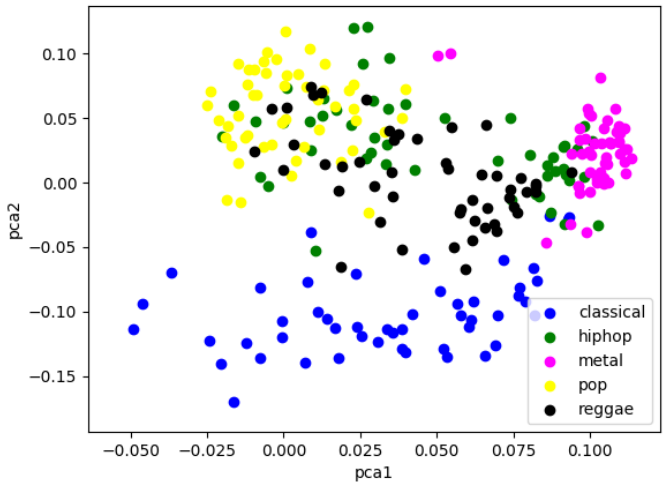
Even without the inclusion of the more distinguishable genres 3-genre groups are somewhat separable.



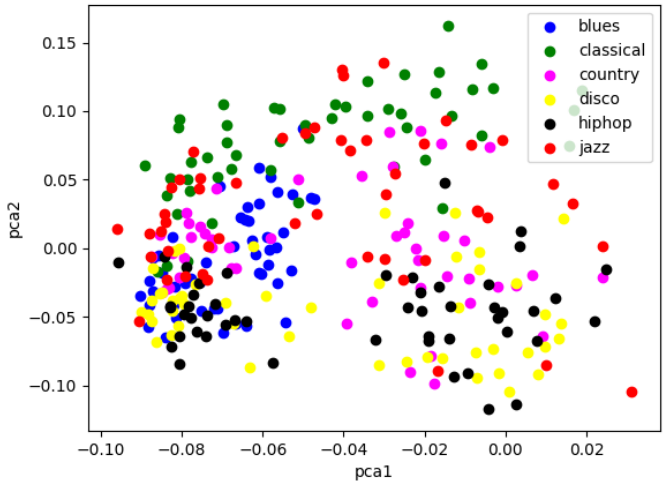


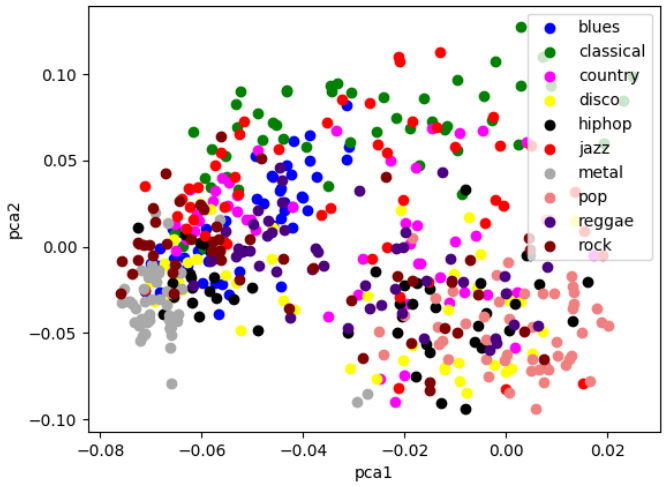
But the situation worsens as we increase the number of genres.





Groups with 6 or more genres are virtually inseparable in PCA plots.





10 genres

Which indicates the classification problem gets significantly harder with more genres.

Artifical Neural network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example.

How humans learn

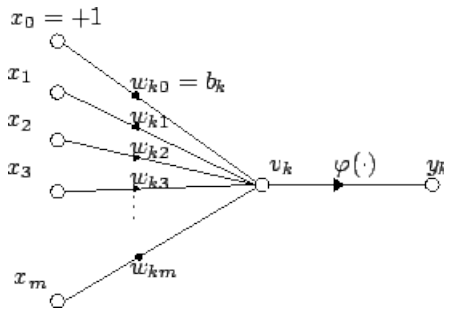
Much is still unknown about how the brain trains itself to process information, so theories abound. In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites. The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurones. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

A simple artificial neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. For a given artificial neuron, let there be $m + 1$ inputs with signals x_0 through x_m and weights w_0 through w_m. Usually, the x_0 input is assigned the value $+1$, which makes it a bias input with $w_{k0} = b_k$. This leaves only m actual inputs to the neuron: from x_1 to x_m. The output of the k th neuron is:

$$y_k = \varphi \left(\sum_{j=0}^{m} w_{kj} x_j \right)$$

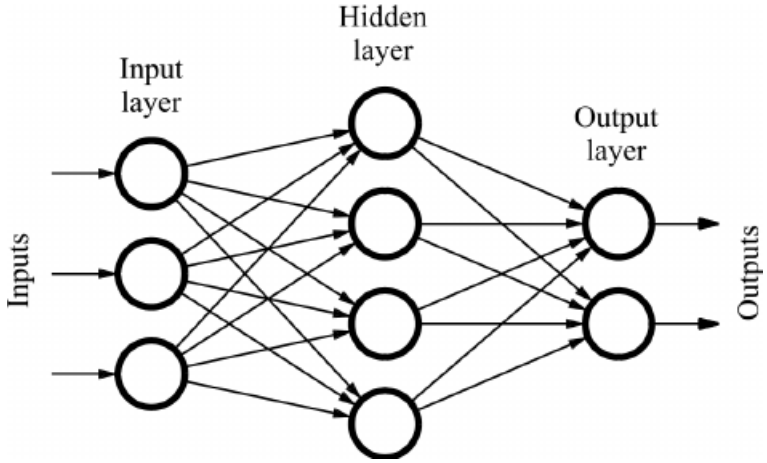
where φ is called an activation function. Activation functions are used to inject non-linearity (and thereby more complexity) into our system. Only nonlinear activation functions allow ANNs to compute nontrivial problems using only a small number of nodes.



Artificial neuron

Layers

ANNs consist of multiple layers, outer layers are called input and output layers and the layers in between are called hidden layers. Each node in a hidden layer is an artificial neuron as in the previous section. Input is fed into the input layer and value y_k is computed for each node v_k in the first hidden layer. These y_k's act as the input layer for the 2nd hidden layer and so on until we finally reach the output layer. This is called a feedforward neural network where layer i treats layer $i-1$ as input layer and layer $i+1$ as the output layer and there are no direct connections between layer i and layer j if i, j are not consecutive integers. More complicated ANNs exist but we shall use a simple feedforward ANN for our experiments.



Feedforward ANN

Setup and methodology

Our feedforward ANN consists of either 2 or 3 hidden layers. We randomly select 50% of the data for training our ANN and rest for testing with equal weight in each genre. We use rectifier as our activation function for hidden layers defined as follows

$$f(x) = \max(x, 0)$$

which is as of 2017, the most popular activation function for deep neural networks[2]. We use softmax activation for our output layer. Softmax is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range [0, 1] that add up to 1.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=0}^{K} e^{z_k}}$$

Our input is layer has 2080 nodes, 1 for each MFCC feature and output layer has n nodes where n is number of genres we're analyzing.

Measure of distance/similarity

We interpret the output vector as a probability distribution. If the output is y then we interpret $y[i]$ as the probability of the current track belonging to genre i according to our current model. If the track actually belongs to genre j then the true set of probabilities are given by $e_j = j$-th standard basis vector for $\mathbb{R}^n$. To understand how far off our model is from reality we compute KL divergence between y and e_j. We add these KL "distance"s across all the training samples and call it our cost function. We apply some optimization algorithm (e.g gradient descent) to minimize this cost function and compute optimal weights and biases for the hidden layers. We run our optimization algorithm for a number of training sessions or epochs and then stop. Now that we have our model we feed our test data into the ANN to see how it performs.

We take the genre corresponding to the largest probability in the output vector for a test track as the prediction for the true genre to compute the accuracy of our computed model.

Results

Below we list classification accuracy for training and test data for groups with different number of genres. These results are best among a few runs of the code with different parameters (e.g number of hidden layers 2 or 3, number of nodes in each hidden layer, learning rate or step length for minimization algorithm, number of epochs) in each case. So it is possible to generate better/worse results with our code.

2-genre groups

	Train	Test
classical, metal	100%	98%
metal, disco	100%	82%
metal, country	100%	95%
classical, disco	100%	95%
classical, hiphop	100%	96%
country, jazz	100%	70%
country, reggae	100%	70%
pop, rock	100%	86%

3-genre groups

	Train	Test
classical, metal, pop	100%	95.33%
classical, country, metal	100%	85.33%
pop, rock, reggae	100%	74.67%
blues, pop, reggae	100%	82%

4-genre groups

	Train	Test
classical, metal, pop, reggae	100%	79.5%
disco, hiphop, jazz, metal	100%	65.5%

5-genre groups

	Train	Test
classical, hiphop, metal, pop, reggae	100%	67.6%
disco, hiphop, jazz, metal, pop	100%	64%

6-genre groups

	Train	Test
classical, blues, country, disco, hiphop, jazz	100%	55.67%
disco, hiphop, jazz, metal, pop, rock	100%	56%

All genres

	Train	Test
all	100%	42.4%

Observations and remarks

1. Not all genres are equally distinguishable.

2. Classification problem gets harder as number of genres increase.

3. Inclusion of more distinguishable genres make classification problem somewhat easier.

4. MFCCs can be used to encode human perception of speech/music.

5. Neural networks can be effective tools in pattern recognition but they lack illuminating interpretation.

6. Increasing number of hidden layers doesn't drastically change the prediction capability of the neural network used in our experiment.

References

[1] European Telecommunications Standards Institute (2003), Speech Processing, Transmission and Quality Aspects (STQ); Distributed speech recognition; Front-end feature extraction algorithm; Compression algorithms. Technical standard ES 201 108, v1.1.3.

[2] Ramachandran, Prajit; Barret, Zoph; Quoc, V. Le (October 16, 2017). "Searching for Activation Functions". arXiv:1710.05941

Acknowledgement

Methodology used here is influenced by the paper "Deep learning for music genre classification" by Tao Feng.