Music Genre Classification

Matthew Creme, Charles Burlin, Raphael Lenain
Stanford University

December 15, 2016

Abstract

What exactly is it that makes us, humans, able to tell apart two songs of different genres? The deduction cannot be trivially made from basic music features, such as BPM, or different pitches. Machine Learning techniques have proved to be able to identify trends from large pools of data, and ultimately classify the music. By finding a way of understanding music on a deeper scientific level, this project aims to classify various music samples into genres.

I. Introduction

Many companies nowadays use music classification, either to be able to place recommendations to their customers (such as Spotify, Soundcloud), or simply as a product (for example Shazam). Determining specific music genres is a first step towards this goal.

In this paper, we will present our successive steps towards building different classifying methods allowing us to identify a specific genre from an initial audio file. We will first describe how we collected data, and why this choice of data was pertinent. Then, we offer a possible conversion of this data into exploitable information, and perform feature selection. We will then progress onto presenting our various algorithms and machine learning techniques used for classification. The final output of our algorithms is the prediction of the genre of each input.

We also quickly diverge into composer classification for classical music. Finally, we present our results that we have obtained while studying this problem.

II. Related Work

Music genre classification is not a new problem in machine learning, and many others have attempted to implement algorithms that delve into solving this problem. Using MFCC's has become a popular way to attack this problem and was implemented by [9] and [10]. Now, [8] implemented the delta and acceleration values of the MFCC's as well, thus increasing the amount of information being garnered from the data.

There are other methods that can be used in order to classify music that were not used in this project such as the Octave-Based Spectral Contrast (OSC) or Octave-Based Modulation Spectral Contrast (OMSC) as presented in [11]

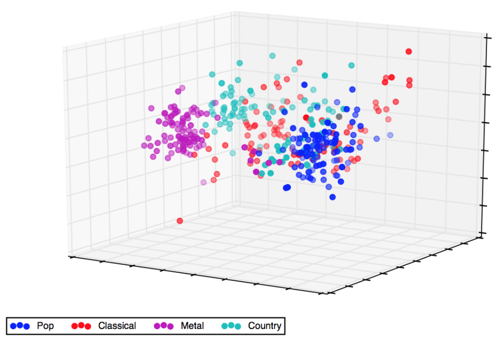
Our software for solving this problem is not available to the public, however [12] shows an open source software that approaches this problem in a similar manner to our project.

III. Gathering data/Feature Selection

We used a dataset that was was taken from the Marsyas software [3]: it is open source and open to anyone to use, in particular in academia, but also in the industry. Our initial data consisted of 1000 .au files (a file extension used for audio files), split in 10 different music genres (Disco, Metal, Pop, Hip Hop, Rock, Blues, Classical, Country, Jazz, Reggae), with 100 samples of 30 seconds for each genre.

Our next step was to find a way to convert this raw data to a format that our machine learning techniques could use. We were referred to the use of [4]: Mel Frequency Cepstral Coefficients. Combining [2] SoX together with James Lyons' [1] MFCC software to obtain our coefficients. In addition, the delta and acceleration values of the MFCCs at each time step, calculated as $\Delta v_i = v_i - v_{i-1}$ and $\text{acc}(v_i) = \Delta v_i - \Delta v_{i-1}$ were used as features. These new features are extremeley important as they measure the transitions between time steps.

Therefore, each training example was represented as a large matrix of roughly 3000 rows (rows correspond to time steps) and 39 columns. Now, most of the algorithms that we used usually treat vectors as inputs, hence we either applied PCA to our matrix, or we flattened the matrix to an extremely large vector, and then used this structure as a training example. Below is a visualization of what a subset of our dataset looks like after applying PCA to reduce the input to three dimensions:



As a slight divergence, we also investigated the topic of composer classification, this time using a different format of files, and hence different features. In order to do this MIDI files were used. The MIDI format contains all the information pertaining to the partition of a piece, which is mainly note pitches and note durations. We used the Python library `music21` to process the raw MIDI files. We chose to isolate the melody which we approximated by the uppermost sequence of notes of a song and we set all the note durations to 1. Hence, a song was there-fore reduced to a sequence of pitches with no timing.

IV. Methods

i. Support Vector Machines

Our first approach was to use SVMs. They are a widely used technique in Supervised Learning and are algorithms that find the best margin in the data. It consists of picking a loss function, and then minimizing our cost function on the training data. Understanding this, the idea is quite simple: consider a specific choice of loss function being the Hinge loss for a single training example x:

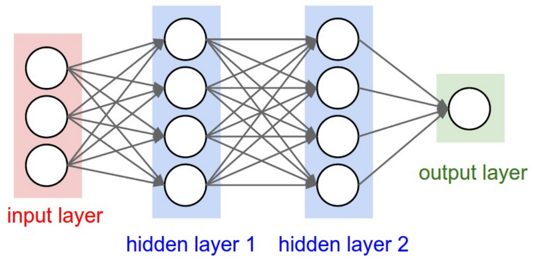
$$l(x) = \max\{0, 1 - x\}$$

The cost function is then straightforwardly computed from there. Running stochastic gradient descent to minimize the cost function, we output a hypothesis function which allows us to make predictions from similarly formatted feature vectors that we would like to test our model on.

SVMs often work better when we use a feature mapping $\phi : S \rightarrow \mathbb{R}^d$ of the data, where S is our feature space, and d is some integer, where often $d > dim(S)$. For our work, we used linear and polynomial kernels of our feature vector.

ii. Neural Networks

Our second approach was the use of Neural Networks. They are very prone to customization, and considering our approach of a rather peculiar feature vector with a rather small data set, this method seemed to be a good choice. The idea is to give our network several inputs (each input is a single feature). Then, we navigate through the network by applying weights to our inputs, summing them, and finally using an activation function to obtain the subsequent nodes. This process is repeated multiple times through the hidden layers, before applying a final activation function to obtain the output layer, our prediction. An example of a neural network with two hidden layers is shown below.



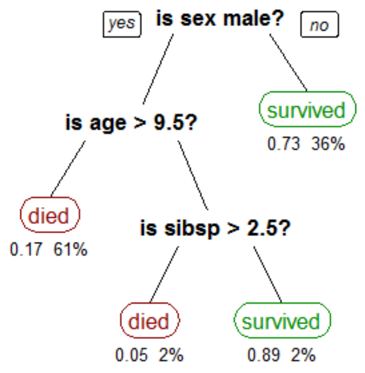
The training takes place on the weights, which are initialized randomly. They are updated by trying to minimize training error using gradient descent, and are used after the training period to try and make predictions.
In our case, we used the softmax function

$$\frac{e^{x_{j,i}}}{\sum_i e^{x_{j,i}}}$$

as our final activation function.

iii. Decision Trees

Decision trees are a rather singular class of classifiers: we would like to construct an optimal tree the nodes consist of question on the input vector (for example: is the third entry greater than 15?). Progressing through the nodes by answering the questions, we eventually reach a leaf, which corresponds to a prediction on the class of the input. An example for a different problem can be seen below.



Building an optimal decision tree can be very costly, because of the exponential nature of the space. However, efficient algorithms have been designed to create very effective trees. We use Hunt's algorithm through a Python package as our training algorithm, which allows us to pick a satisfying tree rather quickly.

iv. K-Nearest Neighbors

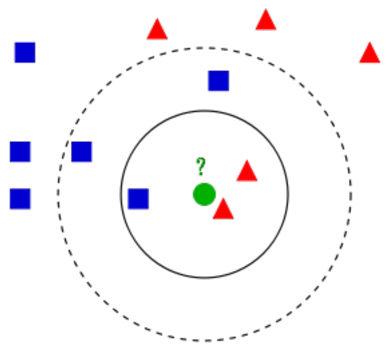
Our last approach was the widely used technique of K-Nearest Neighbors. This is often one of the first techniques applied to classification problems because of its accessibility of implementation. We first applied Principal Component Analysis to our features vector to reduce the dimension of our features space to three, and then ran the K-NN algorithm.

The idea here is to "place" our training set in space, coloring each example using our labels, pick k, and then find the k "nearest" neighbors of the testing example that we are trying to classify. We then make our prediction as the most represented class amongst the neighbors.

The parameters in this model are the distance function, and k. The latter is chosen by training on different values of k, and picking the best one. The distance is often chosen to be the euclidean distance

$$d(x_1, x_2) = ||x_1 - x_2||_2$$

An example of this method can be found below.



v. Composer Classification

We then tried a different approach to classify music. So far, we had selected our features by representing a song using their frequencies. We now consider a song for what it more intrinsically is: a sequence of notes or chords. Our goal is now to find patterns in the partition itself in order to be able to classify the songs into different categories - we will be looking at composers here.

We first needed to transform songs or pieces into their partitions. In order to reach this goal, we used a dataset of MIDI files as described in the "feature selection" section.

From this, the first features we used to classify a song was the distribution of the pitch intervals modulo 12. In practice, this means taking the difference between the pitch numbers of consecutive notes. Since the musical scale is 12-periodic, taking the difference modulo 12 makes a relevant reduction of the feature space. We must also consider the fact that each song is defined using a specific key signature. Therefore, similar notes will be labeled differently as they may come from songs with different key signatures. Considering note intervals instead of notes themselves removes the arbitrary side of the different key signatures. Lastly, for each interval, we keep track of the last k previous intervals up to some limit parameter *depth*, thus making the final feature space $\{0, 1, ...11\}^{\text{depth}}$.

V. Results and Discussion

i. Genre Classification Results

Our dataset comprised of ten different genres, however, we tested our algorithms on classifying amongst four different genres: Classical, Metal, Country and Pop. The reason for doing this was twofold. Firstly, some of the genres in the dataset, such as blues and jazz, are extremely similar to one another and secondly, our algorithms were not as successful as the number of genres being considered increased.

For all of the algorithms, 70 of the 100 songs of each genre were used for training, and the remaining 30 were used for testing. At first, we were extremely worried about the possibility of over fitting, considering that each training example was an extremely large matrix, roughly 3000 by 39. For this reason, we attempted to first apply PCA, with a wide range of components, to our dataset and then apply each of the described algorithms to the compressed dataset. To our surprise, PCA only improved the results of the K-nearest neighbors algorithm while all of the others achieved greater accuracy without the implementation of PCA. Most of our algorithms were extremely accurate on the training set, obtaining accuracies around 95%, raising the possibility of overfitting.

Now, here are the confusion matrices for each of the algorithms used:

Neural Network

Actual / Predicted	Metal	Classical	Pop	Country
Metal	28	0	0	2
Classical	2	25	0	3
Pop	0	0	28	2
Country	1	2	0	27

SVM-Linear Kernel

Actual / Predicted	Metal	Classical	Pop	Country
Metal	29	0	0	1
Classical	6	20	0	4
Pop	0	0	28	2
Country	2	1	2	25

SVM-Polynomial Kernel

Actual / Predicted	Metal	Classical	Pop	Country
Metal	29	0	0	1
Classical	5	19	0	6
Pop	1	0	29	0
Country	1	4	1	24

K-NN with PCA

Actual / Predicted	Metal	Classical	Pop	Country
Metal	28	0	0	2
Classical	7	22	0	1
Pop	0	0	28	2
Country	2	5	0	23

Decision Tree

Actual / Predicted	Metal	Classical	Pop	Country
Metal	27	3	0	0
Classical	1	28	1	0
Pop	0	0	20	10
Country	0	0	10	20

In order to compare the effectiveness of the algorithms, the precision and recall were calculated, along the with the F_1 score, which is calculated as $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. Since this is a multiclass classification problem, the F_1 score shown is the average of the F_1 scores of each genre in the model. Here are the results:

Model	Test-Statistic	Metal	Classical	Pop	Country	F-Score
Neural Network	Recall	93%	83%	93%	90%	0.9
	Precision	90%	93%	100%	79%	
SVM- Linear Kernel	Recall	97%	67%	93%	78%	0.847
	Precision	78%	95%	93%	83%	
SVM- Polynomial Kernel	Recall	97%	63%	97%	80%	0.838
	Precision	81%	83%	97%	77%	
K-NN with PCA	Recall	93%	73%	93%	77%	0.842
	Precision	76%	81%	100%	82%	
Decision Tree	Recall	90%	93%	67%	67%	0.793
	Precision	96%	90%	65%	67%	

Based on the table above, it is evident that neural networks worked the best out of all of the algorithms that we implemented. However, all of the algorithms were still fairly successful classifying the genres considered. In addition, our results compare favorably with the other classifiers that used MFCC's as outlined in the related work section of the report.

ii. Composer Classification Results

Using the previously described 12-dimensional feature space to represent the songs, both SVM and KNN gave relatively poor results. However, classifying a test sample as the training group with the closest average in terms of L_2 norm worked better.
We obtain the following results:

Composer Pair	Beethoven Liszt	Beethoven Mozart	Liszt Mozart
Optimal Depth	2	2	3
Precision	75%	100%	100%
Recall	60%	40%	83%
F-Score	0.67	0.57	0.93

VI. Conclusion, Future Work

We have successfully implemented a piece of software to classify music among genres, which would be a first step towards the construction of a strong architecture that could be applied to recommendations, or music recognition. We observed that the best technique used was Neural Networks, which is probably explainable by the nature of our training examples: very large dimensional continuous vectors. While this choice was our the best approach, we were also rather surprised with the lack of over fitting and the success of the applied techniques.

Using more time, we would have liked to explore other data sets, and cross test our techniques to new testing sets. In addition, we looked at implemented a convolutional neural network. However, we didn't have enough data to make this approach work well. We also only briefly touched on the subject of .midi files, which contain a more analytical interpretation of music as opposed to our physics based understanding, treating it as sound. It would have been interesting to further go down this route, by developing powerful algorithms that could deal with this format.

References

[1] jameslyons *"python_speech_features"*. https://github.com/jameslyons/python_speech_features

[2] SoX *"Sound eXchange Homepage"*. http://sox.sourceforge.net/

[3] Marsyas *"Data Sets"*. http://marsyasweb.appspot.com/ download/data_sets/

[4] Practical Cryptography *"Mel Frequency Cepstral Coefficient (MFCC) tutorial"*. http://practicalcryptography.com/miscellaneous/machine-learning/ guide-mel-frequency-cepstral- coefficients-mfccs/

[5] music21 *"music21 3.1.0"*. https://pypi.python.org/pypi/music21.

[6] M. Nielsen *"How the backpropagation algorithm works"*. neuralnetworksanddeeplearning.com. 2016.

[7] Pedregosa et al. *Scikit-learn: Machine Learning in Python.* JMLR 12, pp. 2825-2830, 2011.

[8] I. Karpov *"Hidden Markov Classification for Musical Genres"*. http://www.music.mcgill.ca/ĩch/classes /mumt614/similarity/GenreHMM.pdf

[9] J. Cast et al. *Music Genre Classification.* http://cs229.stanford.edu/proj2013/FauciCastSchulze-MusicGenreClassification.pdf

[10] M. Haggblade et al. *Music Genre Classification.* https://pdfs.semanticscholar.org

[11] C. Lee et al. *Automatic music genre classification using modulation spectral contract feature.* http://pdlab.csie.chu.edu.tw/

[12] jazdev *genreXpose* https://github.com/jazdev/genreXpose