Extracting Features and Building Learning Machines to Detect Instruments in Digitally Recorded Music

Tim Aris, Jackson Killian, Nick Meyer

Abstract

Automatic instrument recognition can be accomplished by transforming an amplitude signal from a song into many features related to frequencies, pitch and power and feeding these features into a learning machine. We investigated the IRMAS dataset containing ~6,000 three second recordings in which more than one instrument may be playing, but only one is the "lead". To classify clips by lead instruments, we implemented three styles of learning machines that have shown promising results on this task: K-nearest-neighbors, Support Vector Machines, and Neural Networks (deep and shallow). We found that shallow networks performed the best at ~57% accuracy over 11 classes, though the performance of all the machines was heavily dependent on the input feature set. More work guided by audio engineering techniques is needed to further improve accuracy.

Introduction

The exponentially growing body of digital music available online presents a unique set of challenges for those who maintain it. Categorizing music by genre, emotion, and more has implications as basic as optimizing storage and retrieval, to as complex as aiding algorithm-based music recommendation engines (Spotify, Pandora, iTunes, etc.) Identifying the primary instruments present in digital music can help identify the genre or emotional content of the song, but doing this programatically is an ongoing challenge. Herein, we investigate the methods used in a 2012 thesis paper attempting to implement instrument classification in a rich set of live audio recordings. We start by implementing the methods used in the thesis (where possible), then run several different feature extraction and selection methods to attempt to improve on the results.

Background

We used the IRMAS data set [1]. It contains samples of real music which vary in length, source, genre, etc. Each sample is classified with its most prominent instrument. The data set contains 6705 training samples and 2874 test samples with 11 most prominent instruments. The training data samples are 3 seconds each and the test data varies in length. This is acceptable because all of the features either do not depend on the time domain or are aggregate functions such as average and variance. The only paper reference that uses this data is the original PhD. thesis of Ferdinand Fuhrmann [2]. Fuhrmann's work get the best results using SVM, K-NN, and Neural Nets. His accuracies with each are SVM (~63%), K-NN (57.4%), and Neural Net (57.9%). Other works from literature suggest these as best methods for sound classification (with basic learning machines) [3,4,5]. Each of these authors agree that, because of the extreme complexity involved in distinguishing between live instruments in a piece, the most important step for achieving good results with any of these machines is feature extraction and selection.

Methods

As previously mentioned, our dataset is made up of wav files which contain an array of amplitudes over time sampled at 44,100kHz. This in its raw form is difficult to use to discern between instruments. Herein we focus mainly on implementing basic classifier architectures which generally work with feature vectors of a size considerably less than the size of the training sample. Hence the most important step of our work was in extracting features from the wav files that had good classification power. As in typical in the literature, we derived the majority of our features by first converting the data to a frequency space using a Fast Fourier Transform, then computing various features from the resulting curve. We derived several features based on Mel Frequency Cepstral Coefficients (MFCC) using 40 bins and a 0.1 second window [6], basic aspects of the frequency spectra (centroid, spread, energies, valleys, rolloff), and energy inside of bark bands [2]. **Table 1** shows the comprehensive list of features that were used at least once during our experiments. We used python for all of the feature extraction and for constructing all of the learning machines.

Description	Type	Num Features
MFCC	Average	13
MFCC	First Order Difference	13
MFCC	Second Order Difference	13
MFCC	Flattened Correlation Matrix	91
PyAudio[7*]	Average MFCC + Spectral	68
Bark Bands	Average	26

Table 1: Inclusive list of features utilized during experiments. Not all were used at once, and not all were used to obtain our best results.

Having established this list of possible features to work with, we then employed several different techniques for identifying the most important features for classification in order to reduce the number of dimensions.

Univariate Feature Selection: In an attempt to raise the accuracy, we turned to feature selection and elimination [8]. Sklearn's SelectKBest function uses a chi squared method to measure the correlation between features and labels [9]. It reports the k features with the highest correlation. The most impactful features generally were the Mel Frequency Cepstrum Coefficients. Removing the noise generated from unimportant features can improve accuracy. Varying the number of features kept as a hyper-parameter, the neural net was the only learning algorithm found to improve with after feature selection. K nearest neighbors and SVMs both dropped in accuracy if any of the features were eliminated.

Recursive Feature Elimination: This was done using the RFE module in sklearn. RFE works by finding the worst feature for predicting the training data, removing it, then repeating the process until the remaining number of features matches the target number.

Feature Importance: The final feature selection method we employed. Carrying this out is simple using SciKit-Learn's ExtraTreesClassifier, which implements an estimator that fits a given number of randomized decision trees on many sub-samples of the dataset. It uses the average of the tree weights to report the importance of each feature in distinguishing between each of the classes.

SVM

The SVM classifier was implemented using both libsvm and sklearn's SVM packages. An RBF kernel was used, which takes the form $K(x, x') = exp(- \| x - x' \|^2/(2\sigma^2))$. An RBF kernel was chosen because it can handle the nonlinear borders between the instrument classes in the feature space. Other nonlinear kernels were considered, but the RBF is generally considered a standard and reliable starting point [10].

Libsvm and sklearn take the same approach to classifying multiple classes with SVMs: creating many two class SVMs. They give the user the option of creating many one vs. all (OVA) classifiers, matching each class against the rest of the data, or one vs. one (OVO) classifiers, matching every instrument against every other instrument in a pairwise fashion. The SVM with the highest confidence or aggregate confidence (in the case of OVO) gives the final classification. OVA has to create num_classes SVMs and OVO has to create (num_classes) * (num_classes - 1) / 2 SVMs. Experiments revealed the two methods gave exactly the same or extremely similar results.

Features used to get the best results were the same base 68 as described above along with barkbands. Barkbands convert the sounds signal from the time domain to the frequency domain with a fast Fourier transform, and then sums over the power shown in specific frequency ranges, called the bark scale. This is all done as it was in Fuhrmann's [2].

The two hyper-parameters for an RBF kernel are C and gamma. C is the confidence that the training samples are distributed just as the test samples. A high C will give lower training error but the boundary drawn will be prone to overfitting. Gamma affects the range of influence each training point influences. C and gamma were found using a naive box search, as is standard [10] and 10 fold cross validation. The best results were found with C = 5, gamma = .1. A full search is shown in Figure 1.

These features and hyper-parameters gave a test accuracy of 37%. This is low compared to the literature.

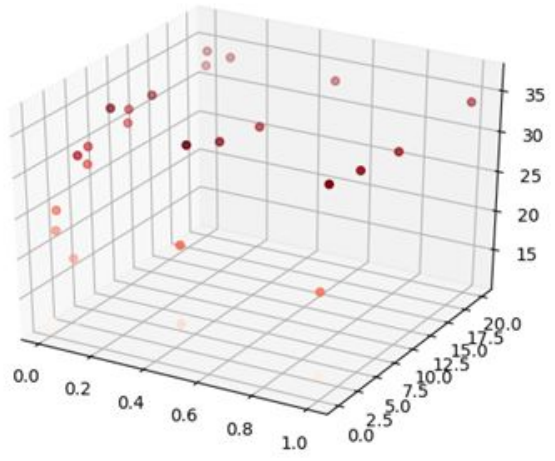

Figure 1. Accuracy over many hyper-parameters. X - Gamma, Y - C, Z - Accuracy %

In an attempt to raise the accuracy, we turned to Sklearn's SelectKBest function uses a chi squared method to find the features that most correlate with the labels. The most impactful features generally were the Mel Frequency Cepstrum Coefficients. Removing the noise generated from unimportant features could have improved accuracy, but in practice it lowered it slightly.

The next attempt with feature selection borrowed ideas from multiclass SVMs. Just as a OVA SVM

makes an SVM for each class and takes the highest confidence, we took the most important features for each class and created a OVA SVM using only those. One SVM for each class trained on that class' specific features. A class' important features were found by passing SelectingKBest between pairs of labels over every pair of labels and aggregating the results. This creates num_claseses multiclass SVMs and num_classes * num_classes two class SVMs. Different feature subsets from a test point's feature set were point into each multiclass SVM, and whichever gave the highest confidence was used. Once again, This practice did not bear better results, generally lowering accuracy a couple percentage points. The most important features varied between labels, but they could have been similar enough to not create the effect we had hoped. Also, in general, removing features from the SVMs always lowered accuracy, so it could be that removing the least important features overshadowed any benefit this method might have had.

K-Nearest Neighbors

K-Nearest neighbors was implemented manually at first, then with sklearn; both in python. The manual implementation used Euclidean distance, while the sklearn implementation used Euclidean, Manhattan, Minkowski, and Chebyshev distances. The methods for calculating distance resulted in the same accuracy (less than 1% difference for most K values), with the exception of Chebyshev, which resulted in a small decrease in accuracy. The Minkowski distance function was used in the end, as that was the default option in sklearn.

Recursive feature elimination was tried in the hopes that one or more of the features was unimportant, or was actively throwing off the knn algorithm. If one of the features was just random noise, then that could result in sound files of the same instrument being far away from each other in the feature-space, and thus decreasing the accuracy of the KNN algorithm. This turned out not to be the case, however, as removing even one feature decreased the accuracy by roughly 5%.

Below is a graph showing the accuracy at various K-values, using the best of the features and models tried (MFCC features, the Minkowski distance function, and not using any feature elimination).

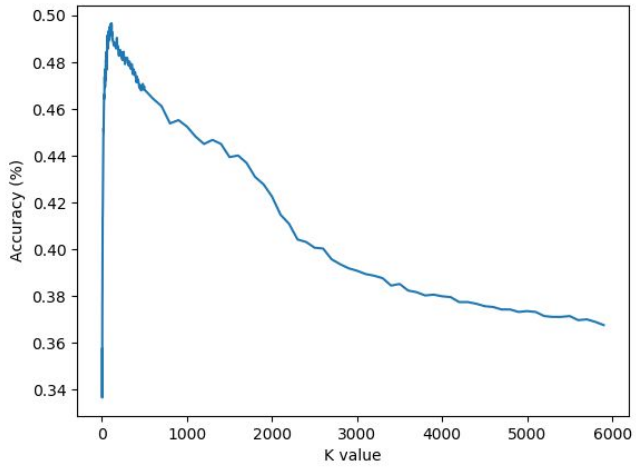

Figure 2. Accuracy of classifications as a function of K in the K-Nearest Neighbors machine.

The best accuracy obtained with k-nearest neighbors was 49.6%, with K values of 105, 110, and 113. Most sound files had more than one instrument playing, so when there were 2 dominant ones, it was considered correct if the model predicted either one. That, and the fact that the most common instrument

in the training data will likely also be the most common instrument in the testing data, is likely why the accuracy is around 37%, rather than 9%, when K is equal to the size of the training data.

Neural Networks

MLP Net: As in the thesis paper, we also implemented a shallow multi-layer perceptron network with one hidden layer (implemented with TensorFlow.) We used a softmax over the output and cross entropy loss with back-propagation to train the network. The learning rate and size of the network were treated as hyperparameters and selected using cross validation for each input feature set. For each set of features, the net was run with its optimal hyperparameters for 2500 epochs. The results are shown in **Table 2** below. Our best combination of features and hyperparameters achieved an accuracy within 0.3% of the literature.

Feature Set	# of Features	Accuracy	Learning Rate	Size of Hidden Layer
MFCC + 1st order	26	32.13%	0.015	20
MFCC + 2nd order	39	30.82%	0.015	35
MFCC - Covariance	91	45.82%	0.003	60
PyAudio Features	68	48.63%	0.006	100
PyAudio top 35%	23	50.25%	0.003	50
PyAudio+Covar top 50%	80	**57.75%**	0.006	100

Table 2: Classification accuracy achieved for 6 different feature sets input to the shallow MLP net. With our best feature set we come **within 0.3% of thesis results for a shallow MLP Net.**

Deep Net: We also attempted to extend our work beyond basic learning machines. We implemented a Deep Neural Network with the following structure from LeCun, et al. [11]: (32x32x1) -> CONV (32x32x32) -> POOL (16x16x32) -> CONV (16x16x64) -> POOL (8x8x64) -> FC (4096x1024) -> FC (1024x13) -> OUT, using a softmax over the output and training with gradient descent. As input we computed a spectrogram over a given number of windows of the sound file, making the number of windows a hyperparameter. Unfortunately, due to limitations to our computational power, we could only create spectrograms with between 1 and 5 windows. Though the accuracy of the model increased steadily with the number of windows, the maximum accuracy achieved was 38.1%. Given the correlation between accuracy and number of windows in the input spectrogram, we likely could have achieved much greater accuracy with this approach if we had access to greater computational resources.

Discussion

We approached a diverse and rich instrument classification dataset using a myriad of feature extraction and selection methods then processed them with common classifying machines. Note that the dataset we use is more challenging than most in this space. Whereas many papers utilize two-class or synthetic music sets, our set contains only live samples from 11 lead instruments. Further, each clip may vary in its

numbers of instruments, volume, recording quality, or genre. We also faced challenges in computational power which restricted us from using important features from literature (Linear Prediction Coefficients) or from extending to more complex classifiers (Deep Nets.) Despite this, we still achieved interesting results, and even virtually matched the accuracy from the thesis baseline for our shallow MLP network. Further, where our results diverge from Fuhrmann's provide interesting insights. It's interesting that our features gave similar results to the literature for the neural net but significantly worse for the SVM. This means that features are of varying utility to different learning algorithms. Our features were of equal utility as the Fuhrmann's when given to a neural net, but much worse when given to an SVM. This means feature importance is affected by learning algorithm and not just the correlation to the labels, and that Sklearn's SelectKBest function does not completely capture the problem. Future work can identify which features are most important for each learning machine, and possibly why.

References

1. "IRMAS: a dataset for instrument recognition in musical audio signals" https://www.upf.edu/web/mtg/irmas
2. Furhmann, F "Automatic musical instrument recognition from polyphonic music audio signals" http://www.dtic.upf.edu/~ffuhrmann/PhD/ffuhrmann_PhDthesis.pdf
3. Herrera P. et al. "Towards instrument segmentation for music content description: a critical review of instrument classification techniques" http://ai2-s2-pdfs.s3.amazonaws.com/3517/43198513dedd4f7d59b8d694fd15caa9a6c2.pdf
4. Liu, J, et al. "SVM-Based Automatic Classification of Musical Instruments" http://ieeexplore.ieee.org/document/5523032/
5. Zhang, T. "Instrument Classification in Polyphonic Music Based on Timbre Analysis" http://www.hpl.hp.com/research/isl/music/itcom01
6. "Mel Frequency Cepstral Coefficient (MFCC) tutorial" http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
7. Tyiannak. "PyAudioAnlysis" https://github.com/tyiannak/pyAudioAnalysis
8. Brownlee, J. "Feature Selection For Machine Learning in Python" https://machinelearningmastery.com/feature-selection-machine-learning-python/
9. "Documentation of scikit-learn 0.19.1" http://scikit-learn.org/stable/documentation.html
10. Hsu, CW, et al. "A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin" https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
11. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.