PROJECT TITLE: A Simple Timbre Detector

Project Members:

Mehmet Alp Şarkışla 2012401075
Nazif Can Tamer 2012401126

Project Members responsibilities:

Our project had two vital parts: feature extraction and implementation of neural network. Alp was mainly responsible with machine learning algorithm and Nazif was responsible for feature extraction. However, there were no clear-cut distinction between our responsibilities as we frequently helped each other and implemented all the algorithms together. Especially the spectral feature extraction algorithm is a product of our extensive collaboration.

The majority of our work relies on libraries and functions built by the open source community. There are plenty of scientific libraries in Python, and we used

- Scipy
- Numpy
- Librosa
- Matplotlib and
- Moviepy
- Sklearn

in our project. Especially, Librosa[7], a musical signal processing library, helped a lot during feature extraction process. Using the Sklearn library we separated training and test portion of our extracted datasets. Alp implemented one-hidden layer neural network function written by Daniel Rodriguez[8]. This is used for classification(training and prediction). Then we check the accuracy of our results with Sklearn accuracy_score.

Problem Statement

Timbre is the quality of sound that distinguishes a particular musical sound from another, even when they have the same pitch and loudness. The American Standards Association definition 12.9 of timbre describes it as, “...attribute of sensation in terms of which a listener can judge that two sounds having the same loudness and pitch are dissimilar,” and adds “...Timbre depends primarily upon the spectrum of the stimulus, but it also depends upon the waveform, the sound pressure, the frequency location of the spectrum, and the temporal characteristics of the stimulus.” (American Standards Association 1960, 45).
Most of the time, timbre is referred to as a subjective quality rather than something that can be measured without human help, unlike pitch and amplitude. The motivation of this project is to automatically detect musical timbre features from various simple sound samples (e. g. monophonic, short, of limited instrument types), and to map them to musical instrument classes as best as we can.

Introduction

To detect objective features of timbre from audio, J. F. Schoutien suggested five acoustic parameters to be related to timbre:

- The range between tonal and noise-like character
- The spectral envelope
- The time envelope in terms of rise, duration, and decay (ADSR-attack, decay, sustain, release)
- The changes both of spectral envelope (formant-glide) and fundamental frequency (micro-intonation)
- The prefix, or onset of a sound, quite dissimilar to the ensuing lasting vibration

We have found several research papers regarding timbre detection. Various acoustical features are analysed to find a viable method to extract timbre information in [1]. MPEG7 descriptors are the most popular and they are based on latest research [1].
In [2], a machine learning approach is presented to extract a musical instrument from a complex music using timbre classification. Influenced by these and considering our limited time to work on at this project, we decided to restrict our feature set so that we only include spectrogram features of the audio.
We have found another useful paper regarding this topic after the proposal: in [3], one can see the the features that can be extracted from audio and to be used in instrument identification process. We mostly stick to the feature space from the paper [3] and implemented those to our work.

Methodology

The methodology almost remained to be the same with the proposal. However, for the classification part we used a feedforward neural network, which is an addition to project

proposal.

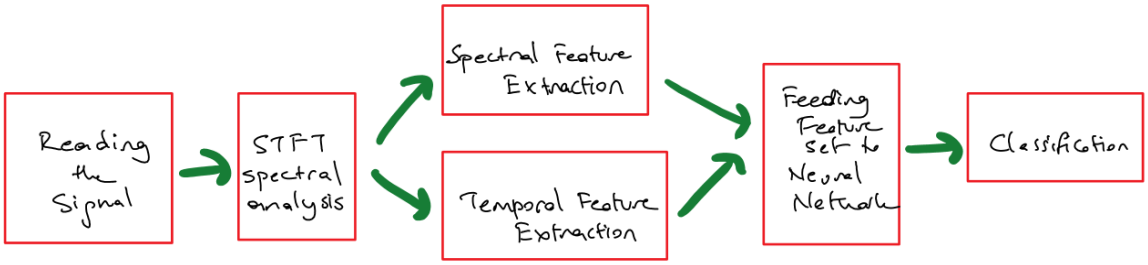

Figure 1 : Methodology

1. We first took short time fourier transform of the input signal, with the fft length of 2048
2. Then we analyzed the fourier spectrum. The main task was to find the harmonic patterns. We accomplished so by
    a. First taking the mean of all the stft(short time fourier transform) windows such that we have only one frame at the end (we acknowledge that this made some calculations redundant; but we landed using this approach due to the added robustness of taking the mean -we also acknowledge that this step prohibited us from recordings that change f0 with time, i.e., the musical phrases)
    b. Then filtering by nearest neighbors the (mean) frequency spectrum to de-noise spectrogram
    c. Applying the peak detection algorithm from Librosa (onset detection)
    d. The peak detection at the previous stage was far from perfect. We took the differential here to find the f0 fundamental frequency (as all the audio has the peaks separated by multiples of f0)
    e. We found the f0 from the difference that is the most frequent.
    f. We removed any frequency less than f0 to be one of peak indices.
    g. And lastly, to find true indices of harmonics -as we lost some precision at the filtering stage-, we looked immediate neighbors of harmonics found after filtering for greater energy.
    h. Relative energies of harmonics are generated by normalizing harmonic energies from energy of f0.

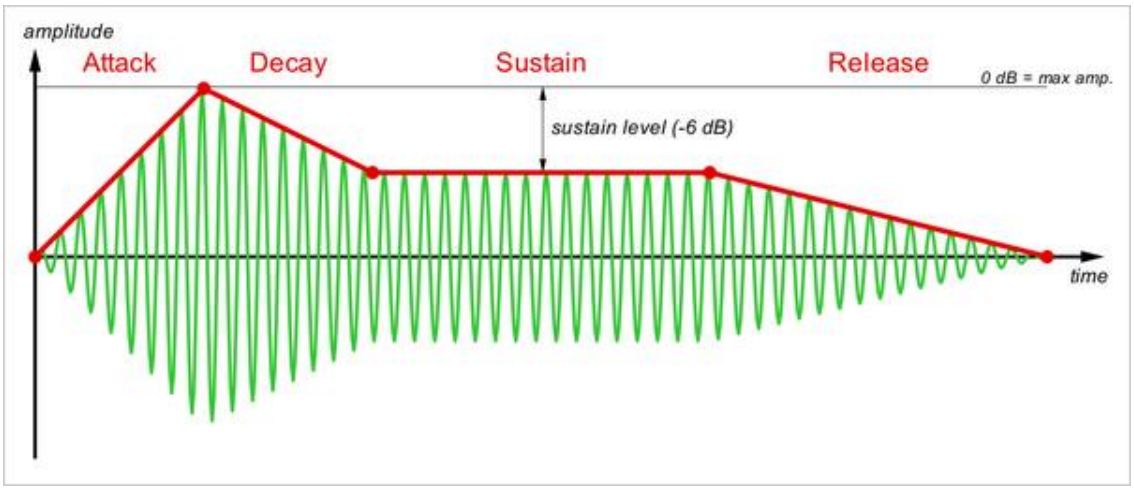

Figure 2 : Envelope

3. The temporal features are also found from spectrogram. We generated signal envelope from the rms value of each stft bin.
    a. We trimmed envelope such that there is no signal component other than the ones belonging to the instrument.
    b. Found attack time, [0:arg(Greatest_Energy)]
    c. Decay time, [arg(Greatest_Energy):arg(Greatest_Energy/4)]
    d. Roll-off rate, energy change rate during decay
    e. Then normalizing everything wrt total signal duration (not in terms of time, but in terms of number of stft windows since we use spectrogram to generate envelope)

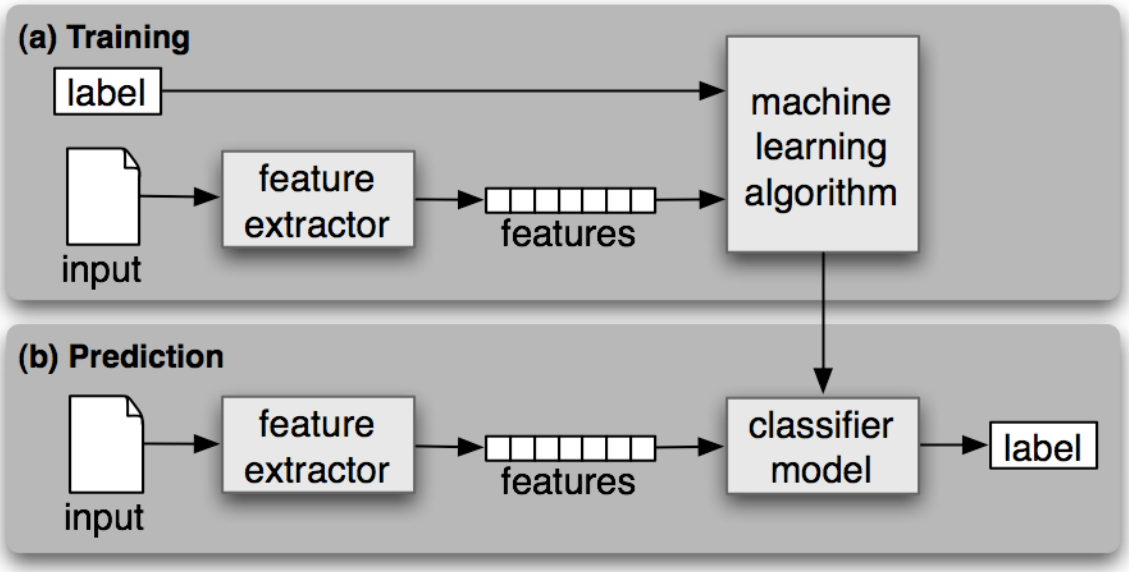

Figure 3 : Machine learning

4. After we get out feature vectors ,supervised classification algorithm called 1 hidden layer Multi-layer perceptron is used.
    a. First we randomly separate samples to be used as a test or train using sklearn cross_validation_train_test_split function.
    b. We have 12 input nodes in total, each sample has 11 features(8 from relative harmonics, others from attack time, decay time and roll-off rate) + 1 bias. And one output number for each instrument (0 for violin,1 for trumpet etc.) This will be fed into machine learning algorithm. Hidden layer size is 3. And activation function is sigmoid.
    c. Algorithm first gives random weights for each product(input to hidden and hidden to output).
    d. Data is forward feeded through the network to generate some (bad, at first) predictions. Then we calculate the error. And using gradient descent optimizer function in sklearn[6]. After many, many...many times weights will be correct and can be used for predictions.
    e. Then we make predictions using test data we separated.
    f. Results will be compared via sklearn accuracy_score

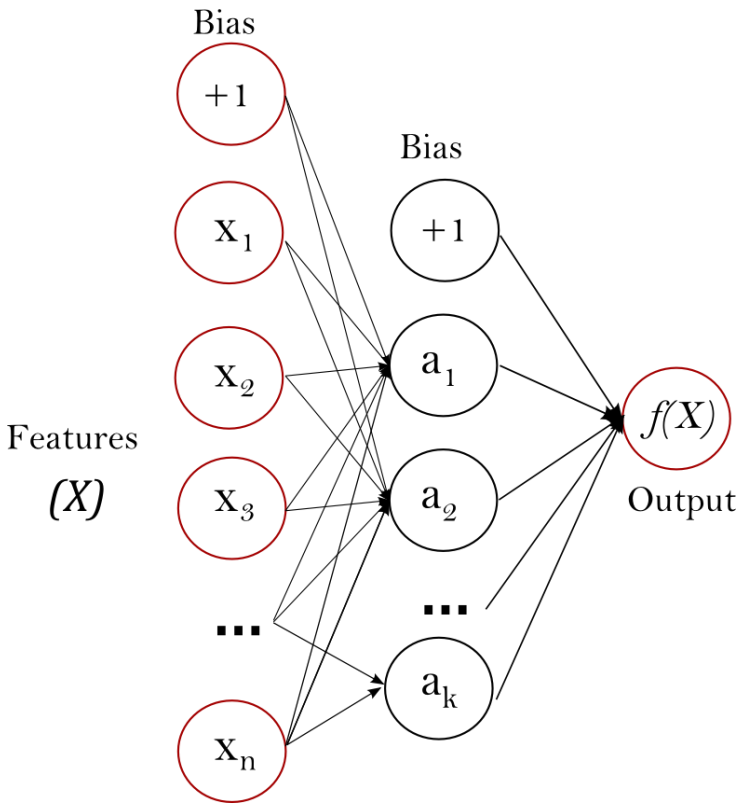

Figure 4 : One hidden layer MLP(Multi-layer perceptron)

Implementation and Simulation Results

We picked Python as the working environment due to its speed compared to MATLAB, and the extensive number of libraries it has in the digital signal processing field. Finding a large and realistic dataset was the core in our project, and we used a dataset from ‘Philharmonia Orchestra’[9], since they had 500 to 1500 samples for each and every instrument used in orchestra. However, the problem with this dataset is that all the recordings were in the proprietary ‘.mp3’ format, and there are no open source libraries to analyze them. We worked around this by installing ‘ffmpeg’ inside Python and this workaround took quite a lot of our time.

We tried various methods for the peak detection, from local maxima detection, to sorting the spectrum energies and then removing the indices around the local maxima. We ended up using the method described in 2.c-2.h in Methodology. Processing the spectrum was, by far, the most time-consuming part of the project.

The audio-processing library Librosa helped us with everything related to the project. We think the best functions in the music processing field reside in this library and recommend that everybody in this field. As you can infer from our code, almost all the built-in functions we used are from this library.

For machine learning we search for the appropriate supervised learning algorithm. At first we tried to use svm(support vector machines) and random forest. Later we have learned it would be better to implement feedforward neural network as a supervised classification algorithm.

Simulation Results(we have used 75% of our samples as training and 25% to test our predictions) :

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
viola finished with size  0
start flute feature extraction
flute :  100 / 478
flute :  200 / 478
flute :  300 / 478
flute :  400 / 478
flute finished with size  478
start violin feature extraction
violin :  100 / 587
violin :  200 / 587
violin :  300 / 587
violin :  400 / 587
violin :  500 / 587
violin finished with size  587
start trumpet feature extraction
trumpet :  100 / 416
trumpet :  200 / 416
trumpet :  300 / 416
trumpet :  400 / 416
trumpet finished with size  416
guitar finished with size  0
start cello feature extraction
cello :  100 / 638
cello :  200 / 638
cello :  300 / 638
cello :  400 / 638
cello :  500 / 638
cello :  600 / 638
cello finished with size  638
horn finished with size  0
saxophone finished with size  0
accuracy:  0.298113207547

Process finished with exit code 0
```

For flute, violin, trumpet and cello; we got only 30% accuracy.

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
start viola feature extraction
viola :  100 / 703
viola :  200 / 703
viola :  300 / 703
viola :  400 / 703
viola :  500 / 703
viola :  600 / 703
viola :  700 / 703
viola finished with size  702
flute finished with size  0
start violin feature extraction
violin :  100 / 587
violin :  200 / 587
violin :  300 / 587
violin :  400 / 587
violin :  500 / 587
violin finished with size  587
trumpet finished with size  0
guitar finished with size  0
start cello feature extraction
cello :  100 / 638
cello :  200 / 638
cello :  300 / 638
cello :  400 / 638
cello :  500 / 638
cello :  600 / 638
cello finished with size  638
horn finished with size  0
saxophone finished with size  0
accuracy:  0.553941908714

Process finished with exit code 0
```

For violin, viola and cello accuracy was 55%. We surprised seeing this result since it seemed like the algorithm detected instruments in same instrument-family (strings) better than quite distinct instruments.

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
viola finished with size  0
flute finished with size  0
violin finished with size  0
start trumpet feature extraction
trumpet :  100 / 416
trumpet :  200 / 416
trumpet :  300 / 416
trumpet :  400 / 416
trumpet finished with size  416
guitar finished with size  0
cello finished with size  0
start horn feature extraction
horn :  100 / 587
horn :  200 / 587
horn :  300 / 587
horn :  400 / 587
horn :  500 / 587
horn finished with size  587
start saxophone feature extraction
saxophone :  100 / 406
saxophone :  200 / 406
saxophone :  300 / 406
saxophone :  400 / 406
saxophone finished with size  405
accuracy:  0.75

Process finished with exit code 0
```

For trumpet, horn and saxophone the accuracy was 75%. Again, these are in the same family but this time their features are distinct to human ear, too. So the fact that it worked better in this case made sense. Yet it was impossible to answer why the accuracy is more than three-folds of the first run.

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
viola finished with size  0
start flute feature extraction
flute :  100 / 478
flute :  200 / 478
flute :  300 / 478
flute :  400 / 478
flute finished with size  478
start violin feature extraction
violin :  100 / 587
violin :  200 / 587
violin :  300 / 587
violin :  400 / 587
violin :  500 / 587
violin finished with size  587
trumpet finished with size  0
guitar finished with size  0
cello finished with size  0
start horn feature extraction
horn :  100 / 587
horn :  200 / 587
horn :  300 / 587
horn :  400 / 587
horn :  500 / 587
horn finished with size  587
saxophone finished with size  0
accuracy:  0.421307506053

Process finished with exit code 0
```

For flute, violin and horn; the accuracy fall to 42%. This was the time we understood there was a problem with flute data samples. When there is flute in our simulation, the accuracy decrease significantly.

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
viola finished with size  0
flute finished with size  0
start violin feature extraction
violin :  100 / 587
violin :  200 / 587
violin :  300 / 587
violin :  400 / 587
violin :  500 / 587
violin finished with size  587
trumpet finished with size  0
guitar finished with size  0
cello finished with size  0
start horn feature extraction
horn :  100 / 587
horn :  200 / 587
horn :  300 / 587
horn :  400 / 587
horn :  500 / 587
horn finished with size  587
start saxophone feature extraction
saxophone :  100 / 406
saxophone :  200 / 406
saxophone :  300 / 406
saxophone :  400 / 406
saxophone finished with size  405
accuracy:  0.79746835443

Process finished with exit code 0
```

When we omit flute and used saxophone instead, we reached 80% accuracy, which proves that the problem is indeed with flute samples.

```
C:\Users\alpsark\Anaconda3\python.exe D:/Projects/Pyton/DspProject/main.py
lets go
viola finished with size  0
flute finished with size  0
start violin feature extraction
violin :  100 / 587
violin :  200 / 587
violin :  300 / 587
violin :  400 / 587
violin :  500 / 587
violin finished with size  587
start trumpet feature extraction
trumpet :  100 / 416
trumpet :  200 / 416
trumpet :  300 / 416
trumpet :  400 / 416
trumpet finished with size  416
guitar finished with size  0
start cello feature extraction
cello :  100 / 638
cello :  200 / 638
cello :  300 / 638
cello :  400 / 638
cello :  500 / 638
cello :  600 / 638
cello finished with size  638
start horn feature extraction
horn :  100 / 587
horn :  200 / 587
horn :  300 / 587
horn :  400 / 587
horn :  500 / 587
horn finished with size  587
start saxophone feature extraction
saxophone :  100 / 406
saxophone :  200 / 406
saxophone :  300 / 406
saxophone :  400 / 406
saxophone finished with size  405
accuracy:  0.564491654021

Process finished with exit code 0
```

For violin, trumpet, cello, horn and saxophone the accuracy was 56%. We believe this was worse because of the size of the 1-layer perceptron. Afterwards we did research a bit about the layer size and found that the size affects the accuracy. The fact that we used different number of instruments for this run was the possible reason of this error, we think. (We used the same size, 3 for both 3 and 5 different instruments.)

Conclusion

We constructed all the feature sets by ourselves, rather than using the MFCC features; the signal processing part of this project was indeed hard and time consuming. Also, we believe, the fact that we implemented the classifier with neural network was a differentiator for this project. Believing that we did a good job, we also acknowledge our two main mistakes:

1. If we had used a bigger feature vector, the accuracy would have increased. However, we couldn't -since we struggled understanding the concept of spectral centroid and didn't have enough time to implement it. In our reference[3], you can see there are 23 possible features to be used in timbre detection and we did use only -the most important- 11.
2. Every time we run the simulation, we constructed the feature vectors from scratch. This is why all of our simulations lasted more than 30 minutes. We acknowledge that if we extract the features beforehand and store the vectors before simulation, we would have find the sweet spot of hidden layer size for our feature set. We used 3, and we believe a more fine-tuned layer size can increase the accuracy.

Given the size of the feature set was only 11, we think the accuracies were indeed high. We believe the relative energies of harmonics and the attack time, decay time, roll of rate are important features of timbre.

References

[1] *Zhang, Xin, and W. Ras Zbigniew. "Analysis of sound features for music timbre recognition." 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07). IEEE, 2007.*

[2] *Park, Sang Hyun. "Musical Instrument Extraction through Timbre Classification." NVIDIA Corporation Santa Clara, CA 95050.*

[3] *Bhalke, D. G., et al. "SPECTROGRAM BASED MUSICAL INSTRUMENT IDENTIFICATION USING HIDDEN MARKOV MODEL (HMM) FOR MONOPHONIC AND POLYPHONIC MUSIC SIGNALS." Acta Technica Napocensis 52.2 (2011): 1.*

[4] *Zhang, Cynthia Xin, and Zbigniew W. Ras. "Differentiated harmonic feature analysis on music information retrieval for instrument recognition." GrC. 2006.*

[5] *Town, Stephen M., and Jennifer K. Bizley. "Neural and behavioral investigations into timbre perception." Frontiers in systems neuroscience 7 (2013).*

[6] *http://scikit-learn.org/stable/modules/neural_networks_supervised.html*

[7]http://nbviewer.jupyter.org/github/bmcfee/librosa/blob/master/examples/LibROSA%20demo.ipynb

[8]http://danielfrg.com/blog/2013/07/03/basic-neural-network-python/

[9]http://www.philharmonia.co.uk/explore/sound_samples