**A10(c): Sound and music description, revisited**

**Question 1: Downloading sounds**

I chose not just ten, but all sounds from: violin, guitar, bassoon, trumpet, clarinet, cello, naobo, snare drum, flute, mridangam, daluo, xiaoluo, I was curious to experience the limits. I also downloaded a lot of samples because from A9 I know that some sounds might not suite their category well.

I changed the soundDownload.py script from A9 to be able to

- search for packs
- have a common folder name even if the queryName is different
- can preview queries without downloading, and review tags of found sounds
- use quotation and allow parenthesis for boolean searches
- multiple calls do not delete but append to the folder
- ... and other miscellaneous changes, all can be found on gitHub [0]

I downloaded quite a lot of sounds for future selection. To accomplish this task, each sound has its own download script [1]

Each uses a slightly different strategy, please look into each script to see which tags, packs or queries I used. I often generated the values for tags by iteration of useful partials. For example:

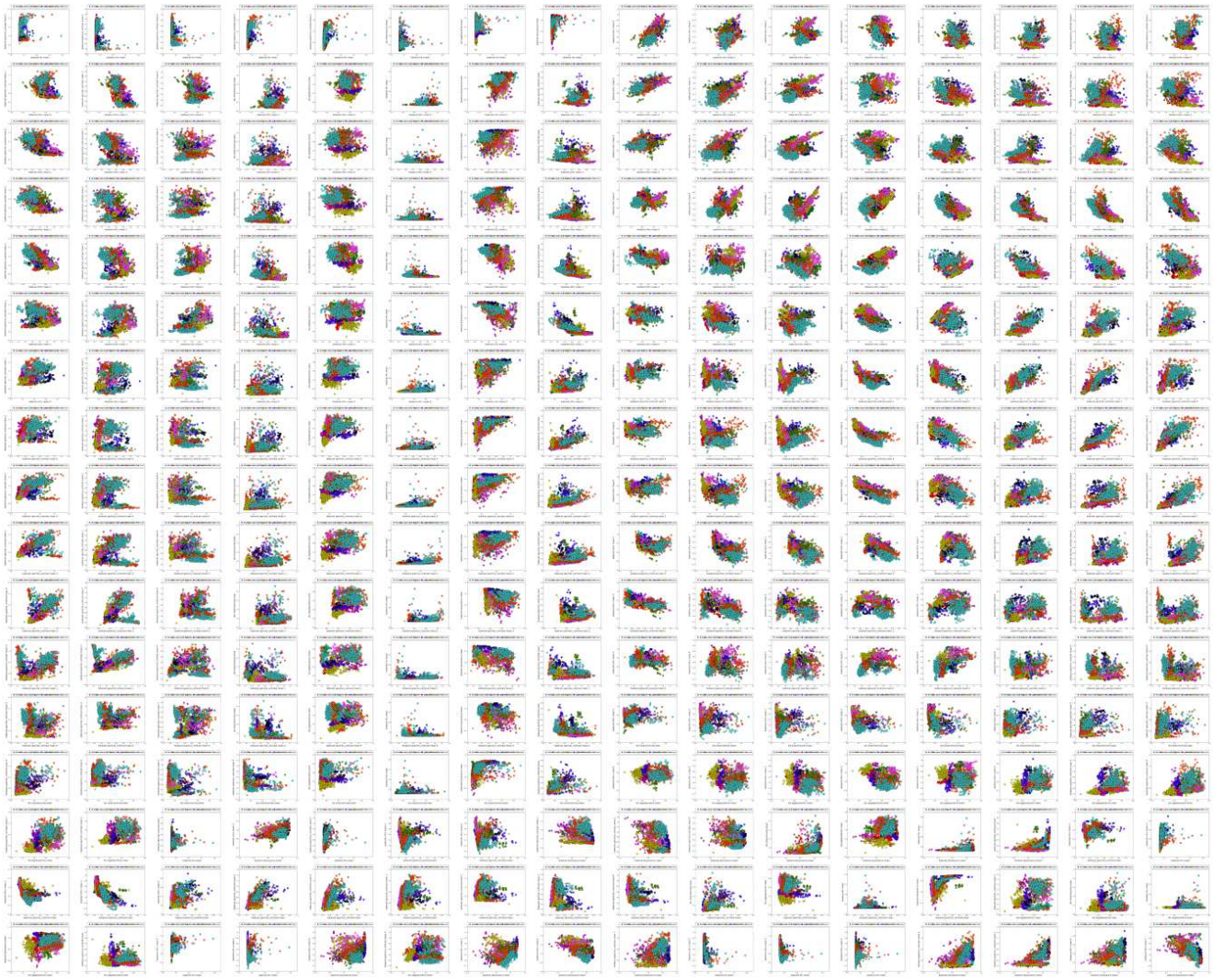
```
nameRange = ["c", "d", "e", "f", "g", "a", "b"]
extRange = ["", "sh"]
soundRange = ["ta", "tha", "tham", "thi", "thom", "cha", "dhin", "dheem", "bheem", "num"]

notes = ["%s-%s%s" % (sound, name, ext) for sound in soundRange for name in nameRange for ext in extRange]

for note in notes:
    params['tag'] = "(mridangam-%s mridangam-stroke-dataset)" % (note, )
    sd.downloadFreesound(**params)
    time.sleep(1)
```

Here [2] I include the links to SoundList.txt files containing the Freesound links to the downloaded sounds.

Now I have a source of the following 1305 sounds of twelve classes: bassoon (79), cello (96), clarinet (136), daluo (50), flute (111), guitar (62), mridangam (174), naobo (46), snare-drum (100), trumpet (277), violin (124), xiaoluo (50). These sounds vary in quality and level and in strength of representing their class. Reductions to the set of sound will be one possible way to tune the model.

**Question 2: Obtaining a baseline clustering performance**

I plotted all combinations of the 17 descriptors we were provided with to get a quick overview. This is done with a simple script [3] and a modification to soundAnalysis.py to save all plots and provice more colors. And a vector graphic tool to arrange the plots (Inkscape). Here is a preview:



I will make a high resolutions version available via my GitHub repository [4].

To get a k-means clustering with the 12 instrument dataset I ran another script [5] that iterates through all possible combinations of descriptors in theoretically arbitrary dimensions. It reports the misses on ten runs and prints descriptor combinations with evolving reduced misses. After finding a set of descriptors which gave good results, I reduced the set to those for faster processing. Tests in four dimensions did not give a bvetter result. These were the results: (The descRange refers to the freesound descriptors in soundAnalysis.pc)

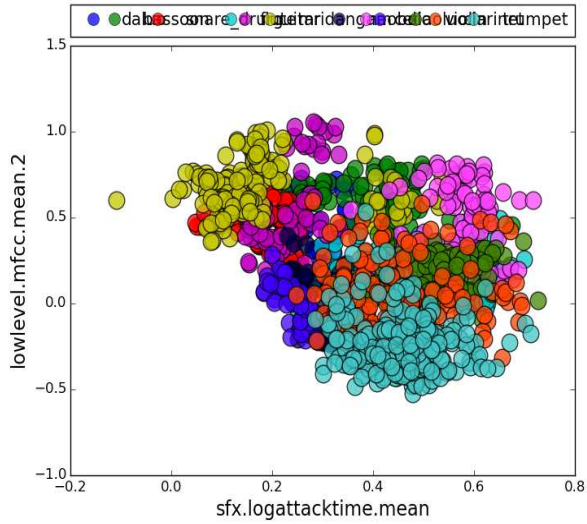
descRange = [2, 3, 10, 13, 14]
retries = 10

[2, 2] misses: 871.3 [ 873. 870. 870. 871. 870. 870. 874. 870. 871. 874.]
[2, 3] misses: 778.3 [ 782. 764. 772. 770. 775. 764. 773. 813. 807. 763.]
[2, 10] misses: 771.1 [ 782. 774. 751. 776. 777. 776. 761. 764. 773. 777.]
[2, 13] misses: 710.6 [ 709. 708. 709. 716. 718. 704. 706. 715. 715. 706.]
[3, 10] misses: 677.8 [ 681. 680. 680. 677. 679. 674. 674. 678. 675. 680.]
[3, 13] misses: 591.8 [ 597. 592. 595. 598. 583. 579. 589. 595. 595. 595.]
---
[3, 3, 13] misses: 577.0 [ 599. 565. 567. 563. 580. 579. 574. 612. 565. 566.]
[3, 10, 13] misses: 562.3 [ 547. 565. 568. 564. 558. 568. 571. 543. 577. 562.]
[3, 13, 14] misses: 510.7 [ 508. 500. 503. 506. 510. 547. 518. 508. 502. 505.]
[13, 3, 14] misses: 506.6 [ 498. 504. 499. 500. 508. 503. 508. 531. 505. 510.]

In two dimensions the best clustering performance with (1305-591.8)/1305 was achived with:

[3,13] sfx.logattacktime.mean, lowlevel.mfcc.mean2

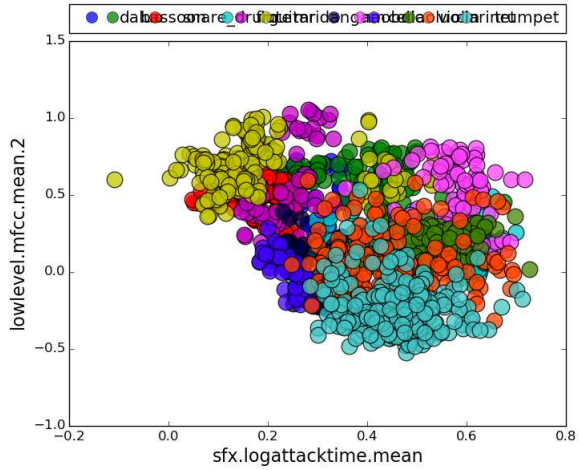
**clustering performance = 54.7%**

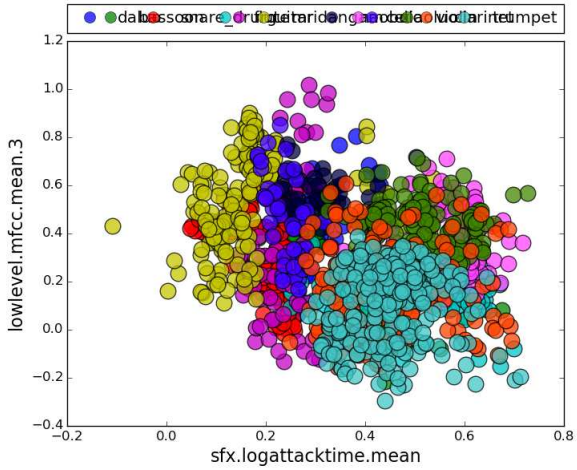



In three dimensions the best clustering performance with (1305-506.6)/1305 was achived with:

[3,13,14] sfx.logattacktime.mean, lowlevel.mfcc.mean2, lowlevel.mfcc.mean3

**clustering performance = 61.3% which is my baseline performance now.**

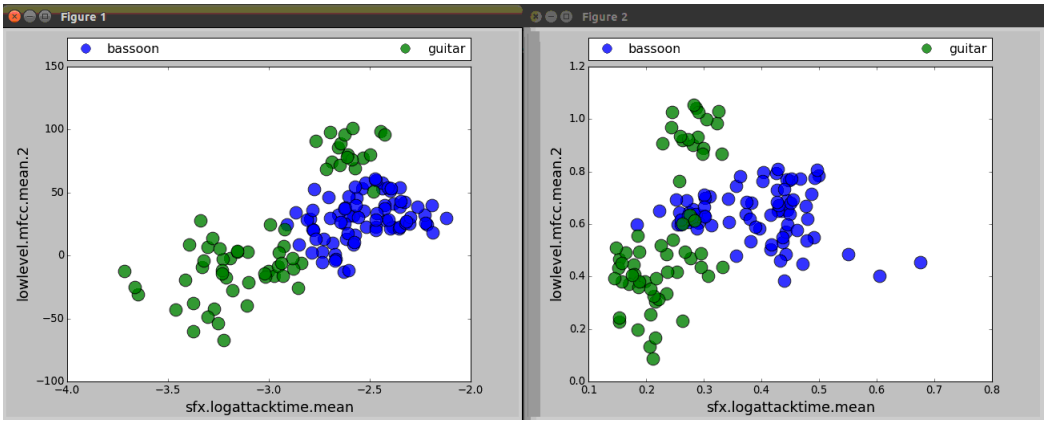



**Question 3: Suggest improvements**

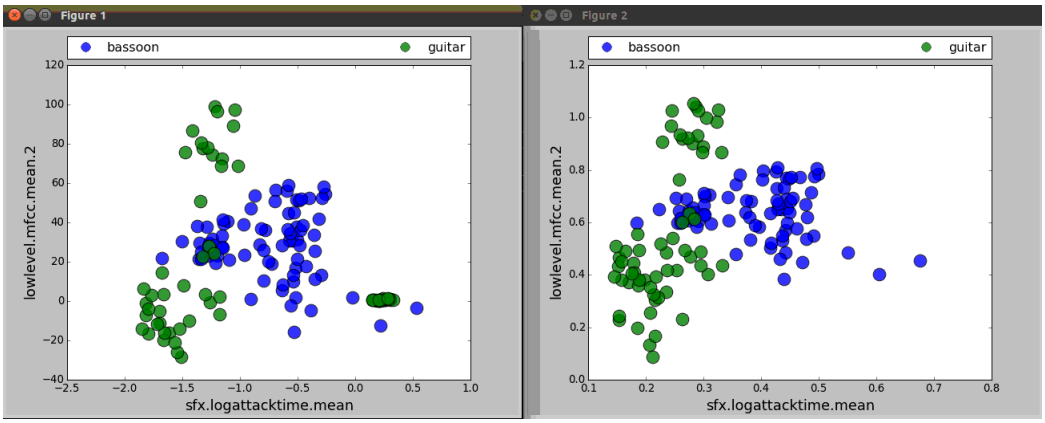
1. Starting from the essentia example in the lectures, I added all descriptors we know from the soundAnalysys.py Freesound script. With those descriptors computed frame by frame I wrote code to "reComputeDescriptors" in which I ran over a copy of all sounds I downloaded. It uses the same json-syntax and filename as the freesound descriptors, so I was able to easily reuse all previous scripts.

The cluster performance was worse then with the Freesound descriptors. The numeric results were also totally different from those of the the freesound descriptors:




left: computed in python with Essentia algorithms - right: Freesound

After discussion in the forum it turned out that the Freesound descriptors are somehow normalized. The question was raised, wether the normalization has an effect on the clustering, so I compared bot freesound normalized with non normalized sounds on two categories:




both freesound descriptors - left: no normalization - right: normalized

Still the non normalized version of the Freesound descriptors are different. While looking for the freesound-extractor implementation in the Essentia sources, I found the following essentia files, containing "ready made" extractors:

> /usr/local/lib/python2.7/dist-packages/essentia/extractor/lowlevel.py
> /usr/local/lib/python2.7/dist-packages/essentia/extractor/sfx.py

Unfortutately they did not work "out of the box", so I first made a local copy [6] and applied some fixes. Now I had a whole bunch of descriptors [7]. I set up my clustering script to evaluate them programmatically. I found the following improvement over the freesound descriptors:

[57, 40, 64] misses: 396.0 [ 396.] [ sfx.effective_duration.mean , lowlevel.mfcc.mean.2 , sfx.tc_to_total.mean , ]

I then wrote a script to combine "essentia" and "freesound" descriptors [8], by prepending another "namespace", and used a selection of descriptors see [9], which proved good in the past and got the following results:

[57, 64, 86] misses: 379.0 [ 379.] [ essentia.sfx.effective_duration.mean , essentia.sfx.tc_to_total.mean , freesound.lowlevel.mfcc.mean.2 , ]

[12, 40, 64, 57] misses: 356.0 [ 356.] [ essentia.lowlevel.spectral_complexity.mean , essentia.lowlevel.mfcc.mean.2 , essentia.sfx.tc_to_total.mean , essentia.sfx.effective_duration.mean , ]

This means clustering in four dimensions, I get a performance of (1304-356)/1304. Here are only 1304 sounds left, because I had to remove one corrupted sound from freesound.

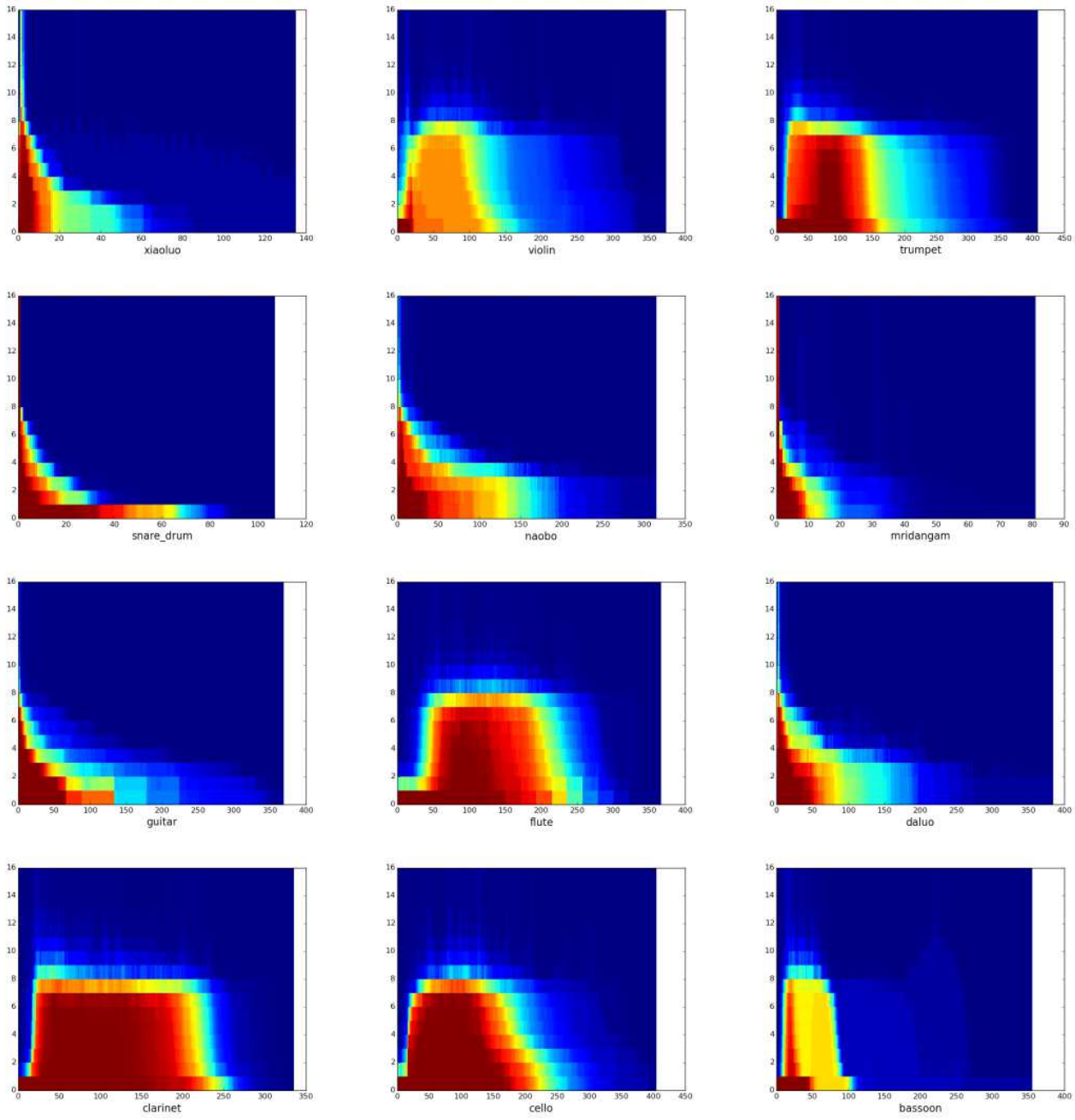
This gives an **improved clustering performance of 73%.**

### 3.2. Computing the descriptors stripping the silences and noise at the beginning/end:

I turn back on my calculation of descriptors. I want to visualize the energy over time on all sounds I used. I thus wrote a script [10] to calculate a histogram over a couple of normalized (eFrame / eMax) energy thresholds for the following values, I chose to have a deeper look at very low and very high energies:

threshold = [eps, 0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 0.98, 0.99, 0.999, 0.9999, 1-eps]

Here is a picture of histograms of all files of all categories:




x-Axis: frame count
y-Axis: threshold slot. Slot 0 shows the length of the files
color coding: blue: 0 or few sounds in this frame@category -> red: many

This shows I can use a threshold of normalized energy per frame of 0.0001.

As a further improvement, I can use different thresholds to compute different sets of descriptors. I was confident that this allows for a even finer clustering. Therefor I started with new namespaces "Energy-ThresHold-Class" (ethc0...15) for the threshold levels shown above. Here is the scipt: [11].

In addition I add "spectral_complexity", because I found out that was a descriptor which provided good results in the past and first implement all threshold slots. That gave me 400 descriptors. Impossible to iterate them in a reasonable time, so they need some reduction. I chose a set of the following, see [12]. Still with my many sounds the brute force algorithm comes to its limits.

For a faster initial evaluation for chosing a further subset, I reduced my sounds to a set of 12*10, just randomly. And quickly calculated a clustering with 29 misses out of 120. The reduced descriptors were used to be run on the large set...

```
------------------------------------------------------------
[0, 0] misses:  886.0  [  ethc1.lowlevel.spectral_centroid.mean ,  ethc1.lowlevel.spectral_centroid.mean ,  ]
[0, 1] misses:  783.0 [  ethc1.lowlevel.spectral_centroid.mean ,  ethc1.lowlevel.dissonance.mean ,  ]
[0, 3] misses:  782.0 [  ethc1.lowlevel.spectral_centroid.mean ,  ethc1.lowlevel.spectral_complexity.mean ,  ]
[1, 3] misses:  752.0  [  ethc1.lowlevel.dissonance.mean ,  ethc1.lowlevel.spectral_complexity.mean ,  ]
[3, 5] misses:  751.0 [  ethc1.lowlevel.spectral_complexity.mean ,  ethc1.sfx.inharmonicity.mean ,  ]
[3, 14] misses:  732.0 [  ethc1.lowlevel.spectral_complexity.mean ,  ethc4.sfx.logattacktime.mean ,  ]
[3, 20] misses:  705.0 [  ethc1.lowlevel.spectral_complexity.mean ,  ethc7.lowlevel.spectral_centroid.mean ,  ]
------------------------------------------------------------
[0, 3, 5] misses:  701.0  [  ethc1.lowlevel.spectral_centroid.mean ,  ethc1.lowlevel.spectral_complexity.mean ,  ethc1.sfx.inharmonicity.mean , ]
[3, 5, 20] misses:  693.0 [  ethc1.lowlevel.spectral_complexity.mean ,  ethc1.sfx.inharmonicity.mean ,  ethc7.lowlevel.spectral_centroid.mean ]
[3, 14, 20] misses:  673.0  [  ethc1.lowlevel.spectral_complexity.mean ,  ethc4.sfx.logattacktime.mean ,  ethc7.lowlevel.spectral_centroid.mean]
------------------------------------------------------------
```

**Conclusion**

Now it's time to get deeper into the algorithms, to understand how the parameters have effect on the resulting descriptors, to discuss and analyse why a certain descriptor is a good choice. My approach to evaluate a huge sets of descriptors on empirical base and then investigate might have taken some time, but it showed what a big effect the the choice of parameters has on the clustering performance.

I'm a bit disappointed about the bad clustering of the threshold-descriptors, and I wonder why the mfcc's don't show up. That definitely needs investigations, but my time is run up. While I write this, I merged my new descriptors together with the freesound and essentia descriptors and see which perform best... the mfcc's did not turn up. That is strange because they did both in the freesound and the essentia namespace.

**Additional thoughts... no time to evaluate...**

Evaluated much more dimensions - 8 to 64 with randomized sets of descriptors:

I was curious if I might "hit" a certain "magic" combination of descriptors, but I didn't. Can refine the algorithm to not use any descriptor twice.

Thoughts about finding the best descriptor combination:

Without going into depth about the theory behind each descriptor, I propose that one decriptor should ideally be able to distinguish two sounds in one dimension. Based on that evaluation, choose descriptors

Thought for another descriptor:

Calculate f0, and do a normalization of all harmonics against f0, so that we get a list of harmonics present, independent from pitch.

Now define pattern in the way: [0, 1, 01, 10, 001, 010, 011, 100, ...] and calculate to which extend it matches the normalized frequency distribution. Each element in the list of patterns is one descriptor.

We might be able to find resonance of certain frequencies wich relate to the physical structure of the instrument. Some common for the class of instrument, some for each individual instrument.

**References:**


[0] modified Freesound download script

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/downloadSounds/soundDownload.py

[1] individual sound download scripts

- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Bassoon.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Cello.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Clarinet.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Daluo.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Flute.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Guitar.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Mridangam.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Naobo.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_SnareDrum.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Trumpet.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Violin.py
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown_Xiaoluo.py

[2] Links to Soundlists

Some of the lists contain quite a lot of duplicates, when searches for different tags hit the same sound. If needed I would have written a script to filter out those duplicates. The sounds and descriptors though exist only once.

- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/bassoon/bassoon_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/cello/cello_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/clarinet/clarinet_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/daluo/daluo_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/flute/flute_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/guitar/guitar_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/mridangam/mridangam_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/naobo/naobo_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/snare_drum/snare_drum_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/trumpet/trumpet_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/violin/violin_SoundList.txt
- https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDown/xiaoluo/xiaoluo_SoundList.txt

[3] Script to produce all plots of descriptor combinations.

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeAnal_All.py

[4] High res version of plot-map

https://github.com/hoinx/sms-tools/blob/02a75955c05480c5557aa82ad71d47df7393f14b/workspace/A10c/plots_17x17_Large.png

[5] k-means clustering brute force script to get the least misses with given descriptors

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeClusterAuto.py

[6] local copy of Essentia descriptors - fixed and mostly working

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/lowlevel.py
https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/sfx.py

[7] Essentia extractors "lowlevel.py" and "sfx.py"

0 : lowlevel.barkbands_kurtosis.mean
1 : lowlevel.barkbands_skewness.mean
2 : lowlevel.barkbands_spread.mean
3 : lowlevel.dissonance.mean
4 : lowlevel.hfc.mean
5 : lowlevel.pitch.mean
6 : lowlevel.pitch_instantaneous_confidence.mean
7 : lowlevel.pitch_salience.mean
8 : lowlevel.silence_rate_20dB.mean
9 : lowlevel.silence_rate_30dB.mean
10 : lowlevel.silence_rate_60dB.mean
11 : lowlevel.spectral_centroid.mean
12 : lowlevel.spectral_complexity.mean
13 : lowlevel.spectral_crest.mean
14 : lowlevel.spectral_decrease.mean
15 : lowlevel.spectral_energy.mean
16 : lowlevel.spectral_energyband_high.mean
17 : lowlevel.spectral_energyband_low.mean
18 : lowlevel.spectral_energyband_middle_high.mean
19 : lowlevel.spectral_energyband_middle_low.mean
20 : lowlevel.spectral_flatness_db.mean
21 : lowlevel.spectral_kurtosis.mean
22 : lowlevel.spectral_pitch_histogram_spread.mean
23 : lowlevel.spectral_rolloff.mean
24 : lowlevel.spectral_skewness.mean
25 : lowlevel.spectral_spread.mean
26 : lowlevel.spectral_strongpeak.mean
27 : lowlevel.zerocrossingrate.mean
28 : lowlevel.barkbands.mean.0
29 : lowlevel.barkbands.mean.1
30 : lowlevel.barkbands.mean.2
31 : lowlevel.barkbands.mean.3
32 : lowlevel.barkbands.mean.4
33 : lowlevel.barkbands.mean.5
34 : lowlevel.barkbands.mean.6
35 : lowlevel.barkbands.mean.7
36 : lowlevel.barkbands.mean.8
37 : lowlevel.barkbands.mean.9
38 : lowlevel.mfcc.mean.0
39 : lowlevel.mfcc.mean.1
40 : lowlevel.mfcc.mean.2
41 : lowlevel.mfcc.mean.3
42 : lowlevel.mfcc.mean.4
43 : lowlevel.mfcc.mean.5
44 : lowlevel.mfcc.mean.6
45 : lowlevel.mfcc.mean.7
46 : lowlevel.mfcc.mean.8
47 : lowlevel.mfcc.mean.9
48 : lowlevel.mfcc.mean.10
49 : lowlevel.mfcc.mean.11
50 : lowlevel.spectral_contrast.mean.0

51 : lowlevel.spectral_contrast.mean.1
52 : lowlevel.spectral_contrast.mean.2
53 : lowlevel.spectral_contrast.mean.3
54 : lowlevel.spectral_contrast.mean.4
55 : lowlevel.spectral_contrast.mean.5
56 : sfx.der_av_after_max.mean
57 : sfx.effective_duration.mean
58 : sfx.flatness.mean
59 : sfx.inharmonicity.mean
60 : sfx.logattacktime.mean
61 : sfx.max_der_before_max.mean
62 : sfx.max_to_total.mean
63 : sfx.strongdecay.mean
64 : sfx.tc_to_total.mean
65 : sfx.temporal_centroid.mean
66 : sfx.temporal_decrease.mean
67 : sfx.temporal_kurtosis.mean
68 : sfx.temporal_skewness.mean
69 : sfx.temporal_spread.mean
70 : sfx.tristimulus.mean.0
71 : sfx.tristimulus.mean.1
72 : sfx.tristimulus.mean.2

[8] Script to combine desciptors, by applying a namespace each:

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joeDescriptorMerger.py

[9] Combined Essentia extractors "lowlevel.py" and "sfx.py" with Freesound extractors

0 : essentia.lowlevel.barkbands_kurtosis.mean
1 : essentia.lowlevel.barkbands_skewness.mean <---- selected
2 : essentia.lowlevel.barkbands_spread.mean <---- selected
3 : essentia.lowlevel.dissonance.mean
4 : essentia.lowlevel.hfc.mean
5 : essentia.lowlevel.pitch.mean
6 : essentia.lowlevel.pitch_instantaneous_confidence.mean <---- selected
7 : essentia.lowlevel.pitch_salience.mean
8 : essentia.lowlevel.silence_rate_20dB.mean <---- selected
9 : essentia.lowlevel.silence_rate_30dB.mean
10 : essentia.lowlevel.silence_rate_60dB.mean
11 : essentia.lowlevel.spectral_centroid.mean
12 : essentia.lowlevel.spectral_complexity.mean <---- selected
13 : essentia.lowlevel.spectral_crest.mean
14 : essentia.lowlevel.spectral_decrease.mean
15 : essentia.lowlevel.spectral_energy.mean
16 : essentia.lowlevel.spectral_energyband_high.mean
17 : essentia.lowlevel.spectral_energyband_low.mean
18 : essentia.lowlevel.spectral_energyband_middle_high.mean
19 : essentia.lowlevel.spectral_energyband_middle_low.mean
20 : essentia.lowlevel.spectral_flatness_db.mean
21 : essentia.lowlevel.spectral_kurtosis.mean
22 : essentia.lowlevel.spectral_pitch_histogram_spread.mean
23 : essentia.lowlevel.spectral_rolloff.mean
24 : essentia.lowlevel.spectral_skewness.mean
25 : essentia.lowlevel.spectral_spread.mean
26 : essentia.lowlevel.spectral_strongpeak.mean
27 : essentia.lowlevel.zerocrossingrate.mean
28 : essentia.lowlevel.barkbands.mean.0
29 : essentia.lowlevel.barkbands.mean.1
30 : essentia.lowlevel.barkbands.mean.2
31 : essentia.lowlevel.barkbands.mean.3
32 : essentia.lowlevel.barkbands.mean.4
33 : essentia.lowlevel.barkbands.mean.5
34 : essentia.lowlevel.barkbands.mean.6
35 : essentia.lowlevel.barkbands.mean.7
36 : essentia.lowlevel.barkbands.mean.8
37 : essentia.lowlevel.barkbands.mean.9

38 : essentia.lowlevel.mfcc.mean.0
39 : essentia.lowlevel.mfcc.mean.1
40 : essentia.lowlevel.mfcc.mean.2 <---- selected
41 : essentia.lowlevel.mfcc.mean.3
42 : essentia.lowlevel.mfcc.mean.4
43 : essentia.lowlevel.mfcc.mean.5
44 : essentia.lowlevel.mfcc.mean.6
45 : essentia.lowlevel.mfcc.mean.7
46 : essentia.lowlevel.mfcc.mean.8
47 : essentia.lowlevel.mfcc.mean.9
48 : essentia.lowlevel.mfcc.mean.10
49 : essentia.lowlevel.mfcc.mean.11
50 : essentia.lowlevel.spectral_contrast.mean.0
51 : essentia.lowlevel.spectral_contrast.mean.1
52 : essentia.lowlevel.spectral_contrast.mean.2
53 : essentia.lowlevel.spectral_contrast.mean.3
54 : essentia.lowlevel.spectral_contrast.mean.4
55 : essentia.lowlevel.spectral_contrast.mean.5
56 : essentia.sfx.der_av_after_max.mean
57 : essentia.sfx.effective_duration.mean <---- selected
58 : essentia.sfx.flatness.mean
59 : essentia.sfx.inharmonicity.mean
60 : essentia.sfx.logattacktime.mean
61 : essentia.sfx.max_der_before_max.mean
62 : essentia.sfx.max_to_total.mean
63 : essentia.sfx.strongdecay.mean
64 : essentia.sfx.tc_to_total.mean <---- selected
65 : essentia.sfx.temporal_centroid.mean
66 : essentia.sfx.temporal_decrease.mean
67 : essentia.sfx.temporal_kurtosis.mean
68 : essentia.sfx.temporal_skewness.mean
69 : essentia.sfx.temporal_spread.mean
70 : essentia.sfx.tristimulus.mean.0
71 : essentia.sfx.tristimulus.mean.1
72 : essentia.sfx.tristimulus.mean.2
73 : freesound.lowlevel.spectral_centroid.mean
74 : freesound.lowlevel.dissonance.mean <---- selected
75 : freesound.lowlevel.hfc.mean
76 : freesound.sfx.logattacktime.mean <---- selected
77 : freesound.sfx.inharmonicity.mean
78 : freesound.lowlevel.spectral_contrast.mean.0
79 : freesound.lowlevel.spectral_contrast.mean.1
80 : freesound.lowlevel.spectral_contrast.mean.2 <---- selected
81 : freesound.lowlevel.spectral_contrast.mean.3
82 : freesound.lowlevel.spectral_contrast.mean.4
83 : freesound.lowlevel.spectral_contrast.mean.5
84 : freesound.lowlevel.mfcc.mean.0 <---- selected
85 : freesound.lowlevel.mfcc.mean.1
86 : freesound.lowlevel.mfcc.mean.2 <---- selected
87 : freesound.lowlevel.mfcc.mean.3 <---- selected
88 : freesound.lowlevel.mfcc.mean.4 <---- selected
89 : freesound.lowlevel.mfcc.mean.5

[10] Script to plot out histograms for all sounds categories:

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joe_Opt2_test_Energy.py

[11] Script to strip silence / noise below an energy threshold:

https://github.com/hoinx/sms-tools/blob/master/workspace/A10c/joe_Opt2_Extract.py

[12] My descriptors with energy-threshold- prefix

0 : ethc1.lowlevel.spectral_centroid.mean
1 : ethc1.lowlevel.dissonance.mean
2 : ethc1.lowlevel.hfc.mean
3 : ethc1.lowlevel.spectral_complexity.mean

4 : ethc1.sfx.logattacktime.mean
5 : ethc1.sfx.inharmonicity.mean
6 : ethc1.lowlevel.mfcc.mean.0
7 : ethc1.lowlevel.mfcc.mean.1
8 : ethc1.lowlevel.mfcc.mean.2
9 : ethc1.lowlevel.mfcc.mean.3
10 : ethc4.lowlevel.spectral_centroid.mean
11 : ethc4.lowlevel.dissonance.mean
12 : ethc4.lowlevel.hfc.mean
13 : ethc4.lowlevel.spectral_complexity.mean
14 : ethc4.sfx.logattacktime.mean
15 : ethc4.sfx.inharmonicity.mean
16 : ethc4.lowlevel.mfcc.mean.0
17 : ethc4.lowlevel.mfcc.mean.1
18 : ethc4.lowlevel.mfcc.mean.2
19 : ethc4.lowlevel.mfcc.mean.3
20 : ethc7.lowlevel.spectral_centroid.mean
21 : ethc7.lowlevel.dissonance.mean
22 : ethc7.lowlevel.hfc.mean
23 : ethc7.lowlevel.spectral_complexity.mean
24 : ethc7.sfx.logattacktime.mean
25 : ethc7.sfx.inharmonicity.mean
26 : ethc7.lowlevel.mfcc.mean.0
27 : ethc7.lowlevel.mfcc.mean.1
28 : ethc7.lowlevel.mfcc.mean.2
29 : ethc7.lowlevel.mfcc.mean.3