1. BIRD CLASSIFIER

This is one exercise from Machine Learning Methods course in last autumn in Tampere University of Technology. The program has been built from scratch by myself, excluding harmonic frequency seeker function.

Assignment:

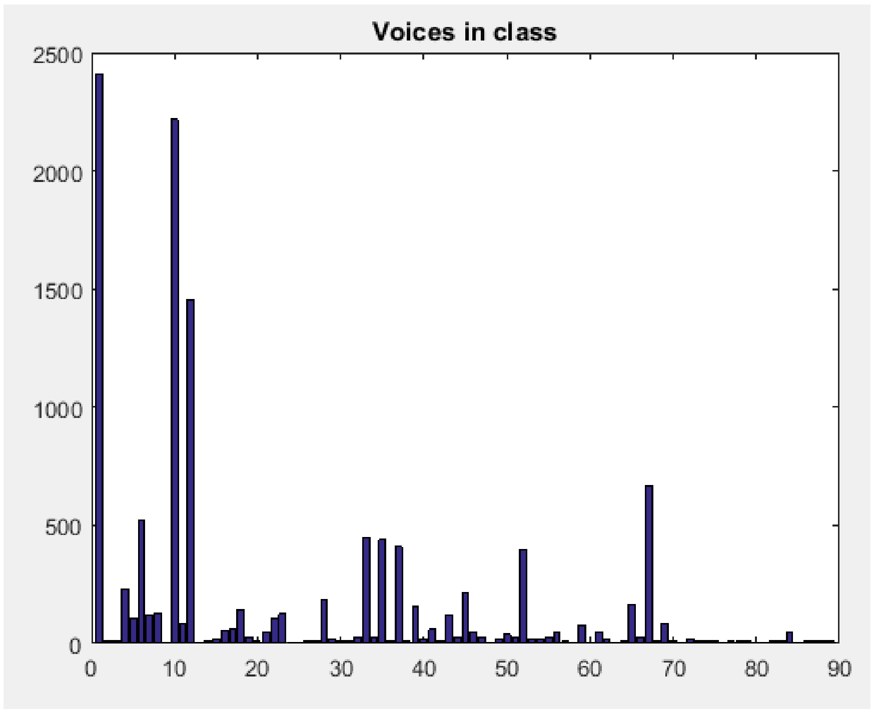
Classify 12000+ voice samples of Fidecula Hypoleuca (kirjosieppo) to their own distinctive categories. Use unsupervised classification, which means that if the sound exists for the first time, it will become a new class in the file folder. If it’s similar to those that exists already in existing folder, move the sound file to the current folder.

Please note that the warping limits (error between two sounds) are not designed to be too tight.

My solution:

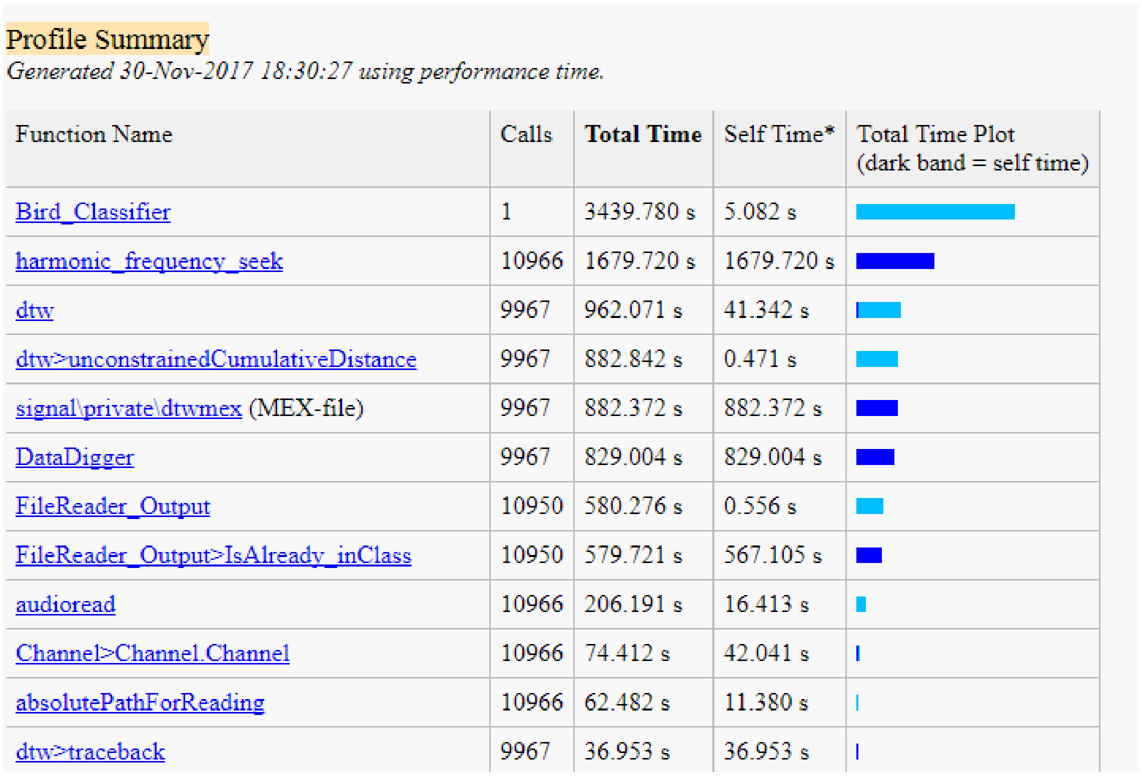
When I started to sketch very first ideas, how to pull off this exercise, I wasn't aware that at the end there would be more than ten A4 size pages of sketching figures and thoughts. Every time different "crazy" idea to solve the problem. It is obvious that many of those attempts failed. Giving up wasn't an option, so I continued implementing and testing my ideas. Few coding implementations and function structures are not as efficient as I wanted them to be, but I didn't have extra time to figure them out. My main goal was to find functional solution. Early on, I realized that the system would contain few distinct parts, so I divided it into sub-functions. My system consists main program Bird_classifier.m, FileWriter.m for file handling, File_Output_Reader.m for file reading and one function called poista_turhat_pilkut.m for removing dots when the new file has been created.

The main idea of my program: The program loops through the list of bird voices. The first voice of every established class is the reference voice for that class. Every new voice candidate is being tested against every reference. If the similarity between reference voice and candidate is below warping limit and it has smallest distance from the voices in reference list, it will be added to that class. Otherwise new class will be established, and voice will be put there. I set warping limit to 7000 and running time of the program was 15 hours total. Here we can notice that about 90 classes where established.



SOMETHING ABOUT EXECUTION TIMES

Since there was more than 12 000 samples and voice comparing appeared to be quite demanding task in a sense of computing. I wouldn't have ever been able to execute my first version of the program in just 15 hours without inspecting bottlenecks from my code. In order to do the inspection, I used MATLAB's tool called Profiler. I tested my code by running through 1000 first samples. There is result from the first test run I did. It took almost an hour to execute the program.


Profile Summary
Generated 30-Nov-2017 18:30:27 using performance time.

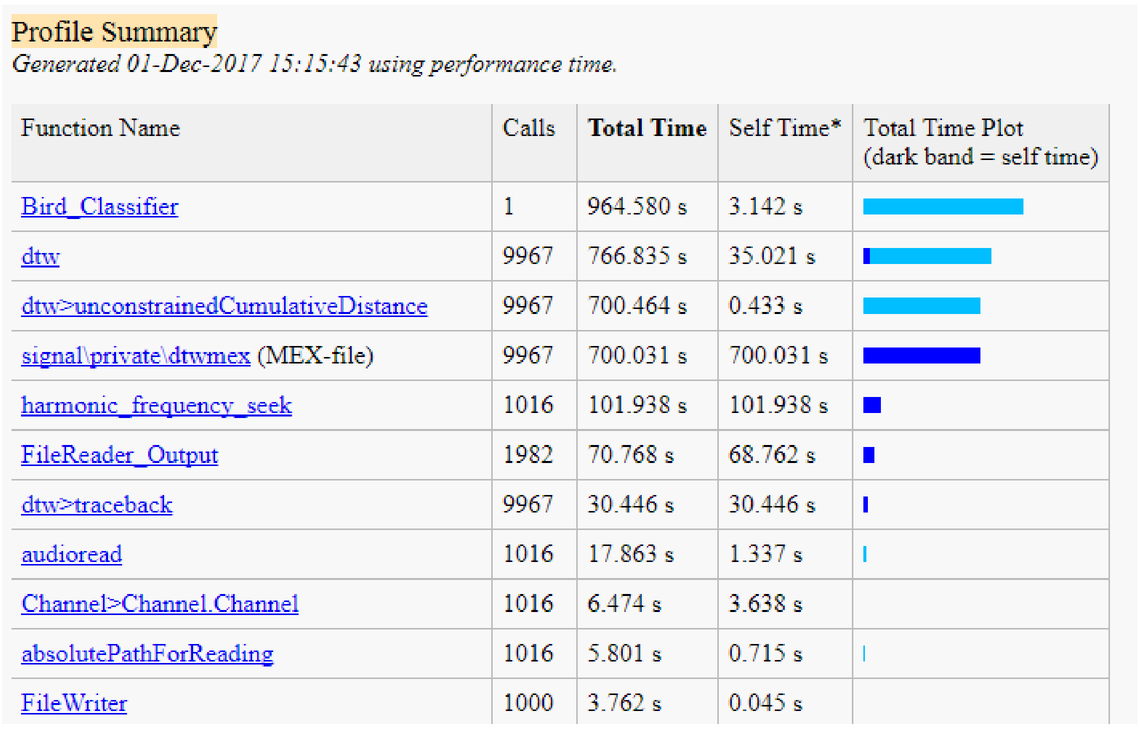
Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
Bird_Classifier	1	3439.780 s	5.082 s	▬
harmonic_frequency_seek	10966	1679.720 s	1679.720 s	▬
dtw	9967	962.071 s	41.342 s	▬
dtw>unconstrainedCumulativeDistance	9967	882.842 s	0.471 s	▬
signal\private\dtwmex (MEX-file)	9967	882.372 s	882.372 s	▬
DataDigger	9967	829.004 s	829.004 s	▬
FileReader_Output	10950	580.276 s	0.556 s	▬
FileReader_Output>IsAlready_inClass	10950	579.721 s	567.105 s	▬
audioread	10966	206.191 s	16.413 s	▬
Channel>Channel.Channel	10966	74.412 s	42.041 s	▬
absolutePathForReading	10966	62.482 s	11.380 s	▬
dtw>traceback	9967	36.953 s	36.953 s	▬

Results were not very promising. There was lot of unnecessary calculations and iteration loops which consumed huge amount of time, for instance DataDigger.

Function was like this:

```
% Tool for digging the dataset
function i = DataDigger(wav_path, haettava)
    for e = 1:length(wav_path)
        if strcmp(wav_path(e).name,haettava) == 1
            i = e;
        end
    end
end
```

Wav_path is the list of all voices. There was something like 111 million calls for this function in 1000 samples test run. Initial idea of this function was to return index of reference voice from the original list. Every time program compared reference voice with some candidate this Datadigger was called. Obviously, that wasn't very clever approach. The solution was to keep track of those references. In other words, to list all of them. Therefore, the reference would be already on the list and it will be unnecessary to loop original list all over again. Another huge improvement was made in function call for harmonic frequency seeker. As we can see from the figure above, it consumed lot of time. I implemented similar strategy than previously, tracking those voice curves. So, reference voice's harmonic curve was calculated merely once and then added to the list.



Profile Summary

Generated 01-Dec-2017 15:15:43 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
Bird_Classifier	1	964.580 s	3.142 s	
dtw	9967	766.835 s	35.021 s	
dtw>unconstrainedCumulativeDistance	9967	700.464 s	0.433 s	
signal\private\dtwmex (MEX-file)	9967	700.031 s	700.031 s	
harmonic_frequency_seek	1016	101.938 s	101.938 s	
FileReader_Output	1982	70.768 s	68.762 s	
dtw>traceback	9967	30.446 s	30.446 s	
audioread	1016	17.863 s	1.337 s	
Channel>Channel.Channel	1016	6.474 s	3.638 s	
absolutePathForReading	1016	5.801 s	0.715 s	
FileWriter	1000	3.762 s	0.045 s	

The figure above shows outcome after I managed to locate those bottlenecks and modify them. Total time was around 2600 second for 1000 samples, I don't know why profiler didn't add those seconds together, like it did before. So, I managed to reduce program's executing time around 75% in 1000 samples. Computing took advantage of parallel computing in MATLAB.