This is meant to be a write up of the variations of the detector and things I have tried as well as the results I have gotten. If you have any questions please email me at [nsj2@rice.edu](mailto:nsj2@rice.edu)

I will be adding tables throughout the document, you can also find these in an excel document (DetectorFinalReportResults.xlsx) . Each table will be its own sheet in the excel document. Look for the document in the folder FinalReport.

If you wish to look at the previous versions of my code:

TheFishSense used to be called mainFiltered and before that main and before that main

In TheFishSense, signal\_detectionFiltered is a local function, but in earlier versions of mainFiltered this was not true, so you may need to look at that too.

TheFishSenseDetectorEvaluator used to be called weekenddrivercalculatornew3

TheFishSenseDriver used to be called CrunchTimeDriver, and before that weekenddriver

SeeLoggedCalls has not had a name change

Wavname2dnumNic has not had a name change

ImportLogasTable has not had a name change

All of the data in this report has been calculated with my most recent copy of TheFishSenseDetectorEvaluator (7/27/17). I used the txt files generated by earlier versions of the detector, and just reprocessed them with the most recent detector.

STEP 1

When I first got the detector it was not compatible with matlab 2013. Originally one of my goals was to keep the detector compatible with 2013 matlab so I could run calculations on other computers in the lab. We also had a goal of 90% accuracy.

At first there were two versions of the code, mainFiltered2016 (the original) with signal\_detection2016 and mainFiltered2013(the one I made to be 2013 compatible) with signal\_detection2013. The part of the code that was not compatible was the builtin matlab function findpeaks. In 2013 it couldn’t handle a time input, so I developed a workaround that would do the time later. After getting my workaround to work I ran step 1.

The detector requires multiple parameters as inputs, the unfortunate thing is that these parameters do interact. Which meant I couldn’t determine what value was best for each parameter individually, I had to consider their interactions. My solution was to just run a ton of trials where I change all of the parameters a lot of different ways and then just look at which way is the best.

\*I had already been through a few iterations by now, but this is the first data I have when I compared what the detector did to the log.

Chunk Duration- scalar in seconds (important for the noise level average)

Threshold – scalar multiplier (multiply by average noise level to set threshold)

Interval – scalar in seconds, interval length is added to the start and end of each call. This means it affects how often the detections will chain together

Peak Distance – a scalar (seconds) that basically says don’t look for peaks within this distance of a peak. There is documentation for this in findpeaks matlab documentation. When this is used matlab finds the biggest peak first then excludes all peaks that are closer than the peak distance, and keeps going from there.

Duration – median duration of detection (median because we have some long periods of noise that were skewing a mean average)

Accuracy – how many logged calls did the detector detect (percent)

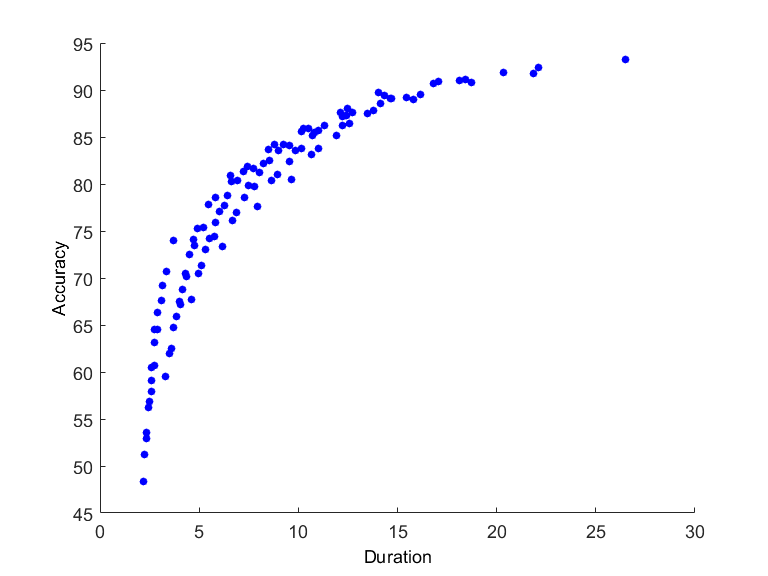
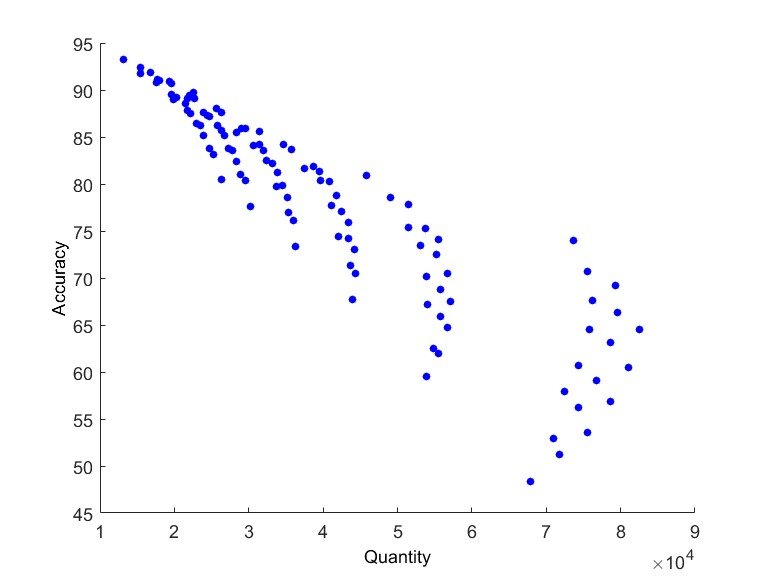
Multiple call % - how many of our detections had more than one logged call in them

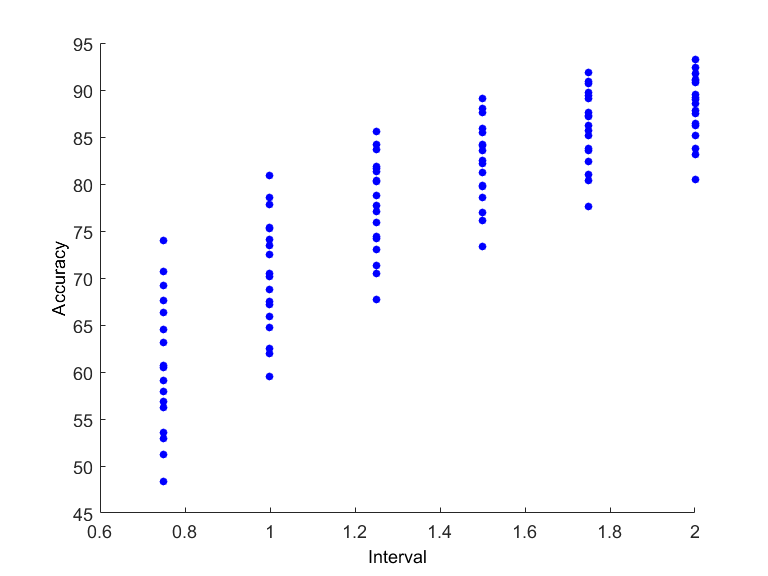
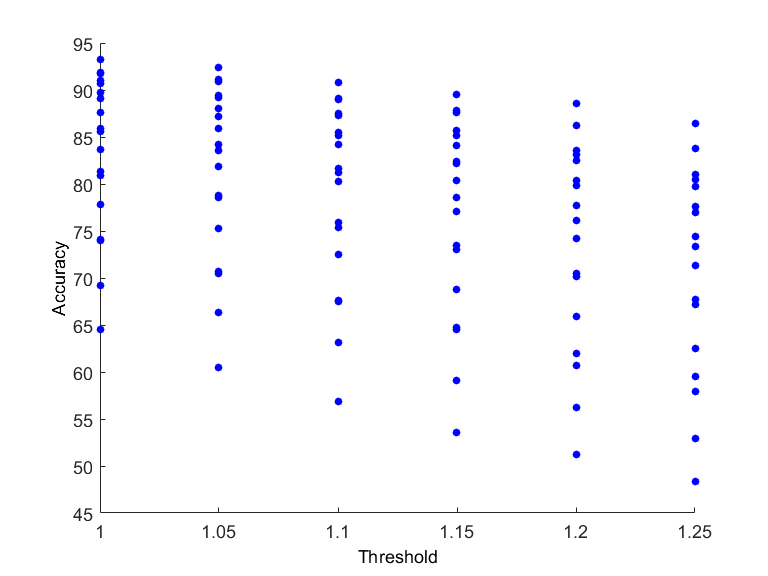
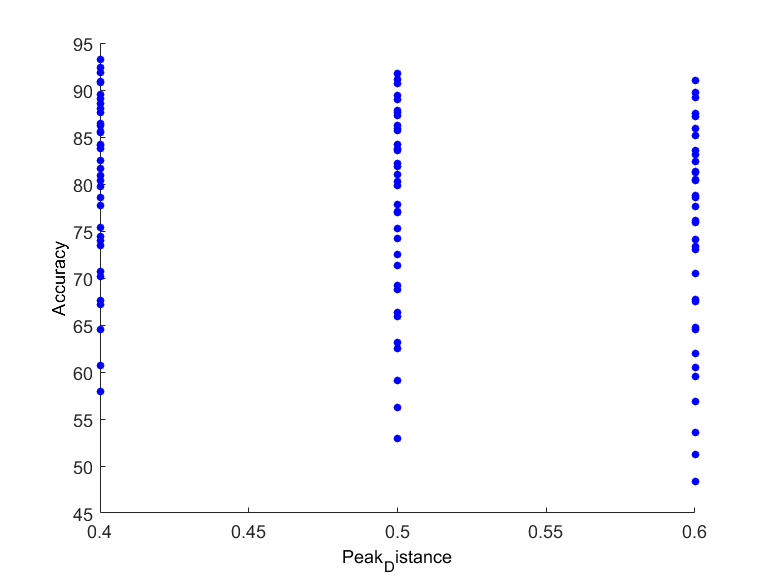
\*\*\* I think these detections were run on main2013.m, but I am not sure. They did do Hilbert transforms on this data

SEE EXCEL SPREADSHEET STEP 1

Peak distance was something that I added to the code, I had hoped that it would make the code run faster because it would take 45min-1hour to process a file (4.5 days). 1hour was a long time to wait to test a new idea. I had hope peak distance would make the code faster by eliminating unnecessary peaks from being detected. But in the end I think peak distance only slowed the code down, because findpeaks would first have to find all peaks then look at the highest and remove.

I plotted some of the trends in the data, and you are going to have to bear with me with these figures. The Accuracy in these figures are calculated by checking if logged starttimes fell into a detection interval. In my updated evaluator I require a detection to fully enclose a logged call to count. That is why the accuracy is higher here.





So these were the results I had from my first big test. This is also where it got weird.

* Best Accuracy with lowest threshold – makes sense
* Best Accuracy with smallest peak distance – kind of makes sense but not too important
* Best accuracy with lowest number of total points – VERY WEIRD!!!
  + Common Sense would tell us that we have a greater chance of randomly getting the logged times if we have more points. But the data is telling us the opposite. I was very confused until I decided to measure the Duration
  + Looking at the Duration graph it because clear,
    - The best accuracy has the longest duration
    - The best accuracy also initially has the most total points, but what happens is there are so many points that more of them join together which results in fewer total points but a longer duration.
    - The best accuracy actually covered a larger duration of time than the lower duration

From now on I started looking at Accuracy and Duration, my new goal because to get 90% accuracy with the lowest duration possible.

At this point I knew I was going to have to change the detector to achieve the results I wanted.

STEP 2

This is where I used TeaganKeiser Energy Detection. Teagan Keiser Energy basically compares the energy of every point to the energy before and after it.

%Teager Energy

teagerdata=zeros(length(data),1);

for i=2:(length(data)-1)

teagerdata(i)= (data(i)^2)-(data(i-1)\*data(i+1));

end

SEE EXCEL SPREADSHEET STEP 2

This was really a proof of concept test.

As you can see the Accuracy isn’t very good, but I chose to run some more intensive tests on the Detector with the Teagan Keiser Energy Detection.

STEP 3

Added a highpass filter

This is a more intensive test of Teagan Keiser Energy and the Detector. I was testing evaluating the Teagan Keiser Detector looking at each sample (what I was previously doing), and averaging every .1 seconds.

SEE EXCEL SPREADSHEET STEP 3 SAMPLES

When doing the samples calculations, it still took about 45-60 min to process the 4.5day long audiofile

This is the data from the averaging over .1 seconds

SEE EXCEL SPREADSHEET STEP 3 SECONDS

I’m not sure why the accuracies here are such garbage. I am pretty sure it was due to some error I made because I chose to stick with averaging over .1 second and the accuracies get much better later.

But one important thing I discovered in this step is that averaging over a second made the computation much faster. It was taking me about 10 min to process 4.5 days of data.

STEP 4

At this point I thought the detector was done and all I needed to do was find the magic combination. I am now using Teager Keiser Energy and averaging over .1 seconds.

I experimented with a moving average filter to smooth the data and it decreased my accuracy so I stopped.

At this point I seem to have solved whatever error I was making earlier.

I also removed the read write read redundancy in the code.

STEP 5

So by this point I added the SeeLoggedCalls function and was able to look at what was happening in the log. I came up with my idea for what I called a DB Filter, but is actually a power threshold filter by looking at the spectrograms. I saw a lot of noise around the call, but this noise was quitter than the actual call. My line of thinking was I want to filter everything out except the call, then the peaks will be as high as possible.

[Insert Spectrogram]