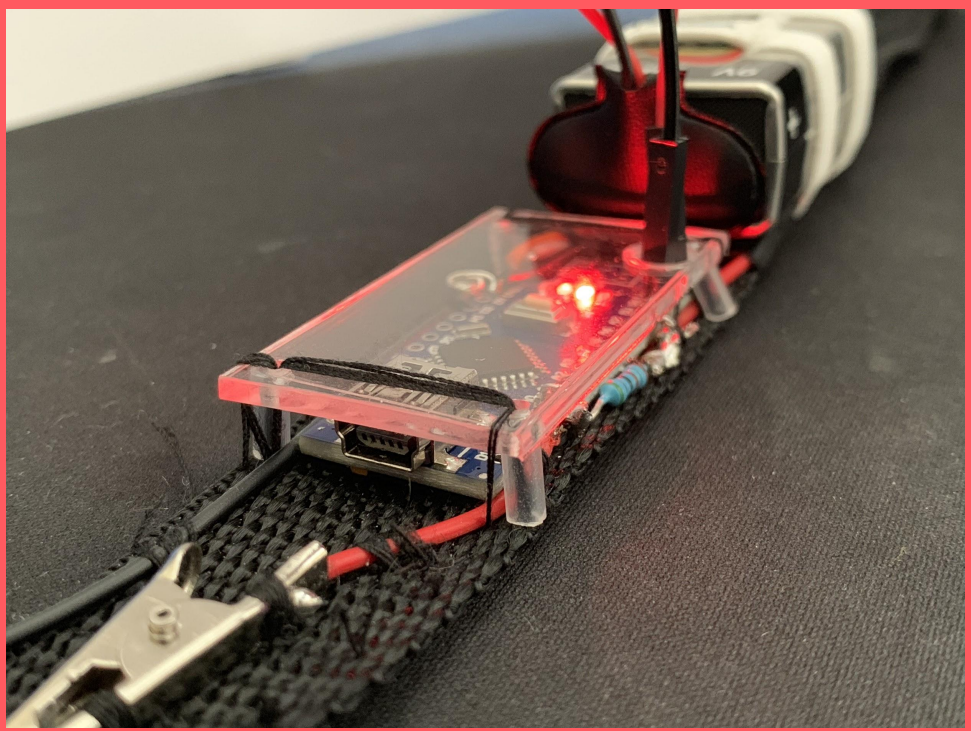
Shashank Venkatramani & Pranav Eranki

Asthma diagnosis and progression tracking through the use of non-intrusive wearable technology and respiratory analysis



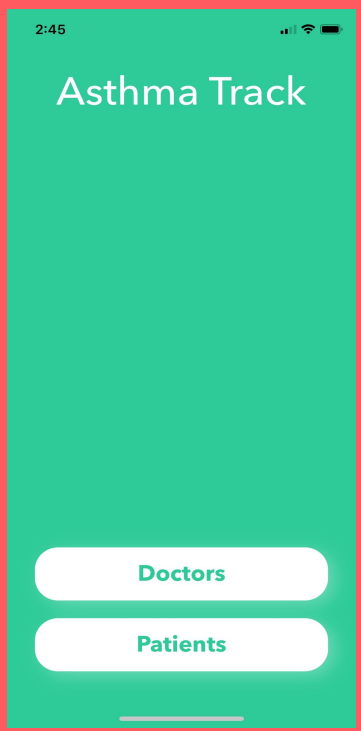


Table of Contents

Table of Contents (cont)

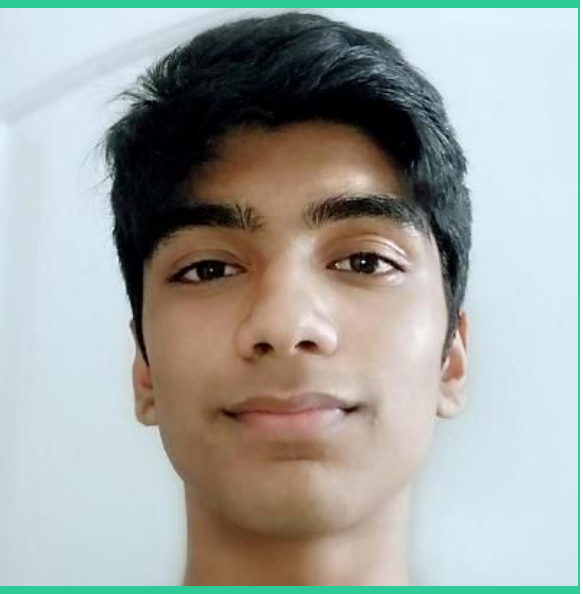
Motivation

Asthma is a condition that when triggered causes the inflammation of airways and increased mucus production causing difficulty of breathing.

It affects 8% of the US population (20 million lives) across all age ranges, leading to 4000 deaths a year, with symptoms also including chest pain, cough, and wheezing.

Currently, the only way to track asthma is to use a peak flow meter, however, it requires the user to blow into the meter and manually track.

Pranav Eranki



10th grader at Cupertino High School

Python / Notebook lead

Shashank Venkatramani

10th grader at Cupertino High School

Hardware / iOS Lead

Meeting & Progress Entries

11/12/2018

Goals

- **Finalize Idea**
- **Work on Research Plan**

Overall:

- **We only managed to finalize idea and do a bit of rudimentary diagramming, so we need to meet again tomorrow**

Notes

We met for 4 hours or so from about 2 to 6 p.m.

Since we were trying to finalize our idea, we spent the first 30 minutes parsing through a list of ideas that we had before.

We decided to settle on an idea which uses a strap-like piece of technology around your chest / stomach that can record how much you breath, how long you breathe for, and essentially use that too graph and determine the user's possible asthma attacks.

We began working opn registration because it was due on the 14th, and we need to finish all the forms and drive down to turn them in by the 13rth, which is the next day.

11/13/2018

Goals

- **Finalize research plan**
- **Drive down to turn in the forms**

Overall:

- **Goals accomplished**
- **Successful day**

Notes

Today, we talked in the beginning of the data, in the morning, then met once during lunch (of school) to finalize what we wanted to accomplish and put in our research plan.

Our research plan mentions many things, the gist of which can be captured by the following lines:

11/13/2018

Goals

- **Finalize research plan**
- **Drive down to turn in the forms**

Overall:

- **Goals accomplished**
- **Successful day**

Notes (cont)

The goal of our project is to design a passive, unintrusive abdominal strap that constantly monitors your breathing, in order to get a thorough reading and progression of the patient's' respiratory patterns and to send this data to your phone, to be processed and presented to the patient and their doctor automatically.

The device will possess the ability to record the breathing rate within the range of 0 to 150 breaths per minute. Another criterion is that the device will be non-intrusive, and must have at least 4 cm of expansion.

11/13/2018

Goals

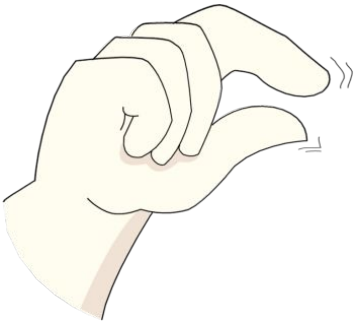
- Finalize research plan
- Drive down to turn in the forms

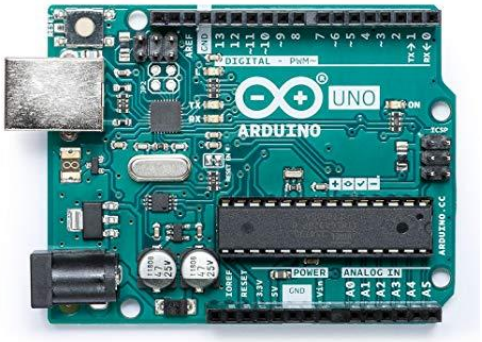
Overall:

- Goals accomplished
- Successful day

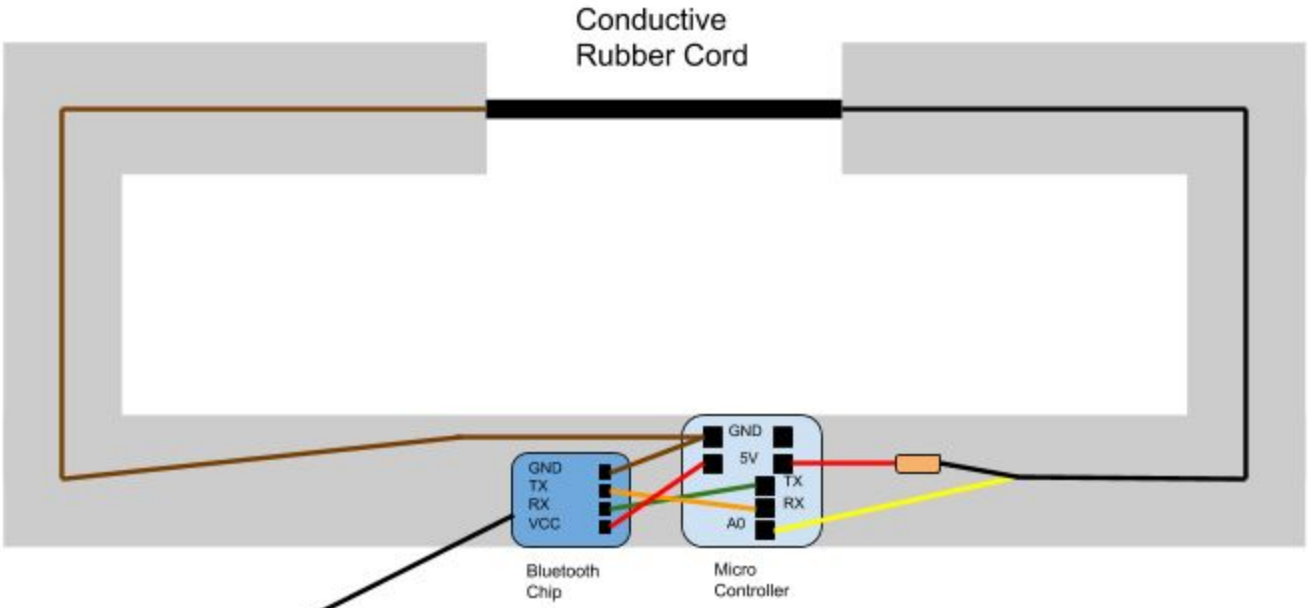
Notes (cont)

The device must weigh less than 2kg and should not cause any major physical discomfort to the patient. The algorithms themselves must not be computationally intensive, as to not slow down the user's device upon which they run, and should run on mid to high tier cell phones.

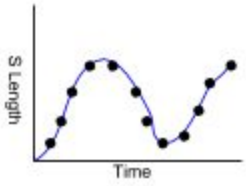




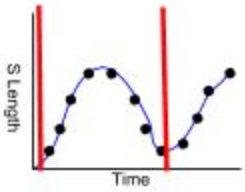
Base Plan as of 11/13/18



Receive length of strap, starting from a calibrated 0 to x (change in user's chest circumference during breathing), in 60 second snapshots at highest reading rate capable by the Microcontroller



Turn data points into a graph over time for later computation



Segment breath graph into individual breaths

11/12/18

Software Algorithm Plan
11/13/18

Construct an instantaneous velocity (derivative) graph out of the graph of the initial breath graph in order to amplify any fluctuations in the data

Then, in order to reveal deeper insights in the data and determine if the graph signifies an asthma attack or not, we would:

Design a custom algorithm implementing features of the graph, including

- Time vs length change
- Inhale vs exhale times
- Slope analysis

or

Write a machine learning algorithm using Convolutional Neural Networks to learn to differentiate between the different instantaneous graphs and automatically classify the graph as 'asthma attack' or 'not asthma attack'

11/13/2018

Goals

- **Finalize research plan**
- **Drive down to turn in the forms**

Overall:

- **Goals accomplished**
- **Successful day**

Notes (cont)

Testing methods:

1: Creating a linear actuator, that stretches out the band to simulate the increasing/decreasing circumference of the human chest cavity during breathing, in order to test the upper and lower limits of the sensor.
2: Test band on ourselves by wearing it throughout the day, checking both the data output and analysis of the device, along with examining any lack of comfort caused by the device.
3: Gathering data by using the device on ourselves for extended time frames to gather data points for standard breathing patterns and breathing rates(will not induce asthma during testing).

11/24/2018

Goals

- Hypothesize Data
- Learn Arduino
- Create Python based helper functions

Overall:

- Goals accomplished
- Successful day

Notes - Software

Created a set of helper functions that:

- Converts a csv file into a set of data from csv to pandas functions to data frame. It uses a custom pandas read function then returns
- Next, we created a function which takes in a text file then takes it into a set of data. It takes the text file path, calls a custom function to get the file at that path, then uses a type analysis to determine if it is a txt file or a csv, thern uses custom panda functions to actually save the dataframe.
- Finally, I wanted a function that would take the set of data and plot it as a matplotlib plot. It takes in the path, gets the data at the point using the above functions,. Then does x y analysis, pots them in a matplotlib figure as a plot., then takes in an output folder and saves the image to that folder under a name which the user specifies.

```
def getType(filename):
    x = filename.index(".")
    return (filename[x + 1:])


def dataToDf(filename, delimiter):
    if (getType(filename) == "csv"):
        return pd.read_csv(filename, header = None)
    else:
        return pd.read_csv(filename, sep = delimiter, header = None)


def dataToImg(dfx, dfy, name):
    fig, ax = plt.subplots( nrows=1, ncols=1 )
    ax.scatter(dfx, dfy, s = 150)
    ax.axis('off')
    fig.savefig(name)
    plt.close(fig)
```

These are the first three helper methods that we created. The first one gets the type of the data file, the second one converts the data file given to a dataframe, and the third one sends the data to a image file and saves it./

11/24/2018

Goals

- Hypothesize Data
- Learn Arduino
- Create Python based helper functions

Overall:

- Goals accomplished
- Successful day

Notes - Hardware

Arduino Uno, breadboard, and dupont wires arrived, and we began testing.

We setup the arduino ide, and was able to run and upload sketches to the board.

Made a quick test program that would light up the on board LED (pin 13), every 500ms, and studied the basics of each pin type.

Made a basic switch circuit to control an led as practice, for drawing, and making a working circuit.

12/01/2018

Goals

- Sequential model for CNNs
- Learn about CV
- Explore ways to detect breathing

Overall:

- Goals accomplished
- We could have spent more time learning about how CNNs work, but we began to ran out of time and set to building the basics of the model

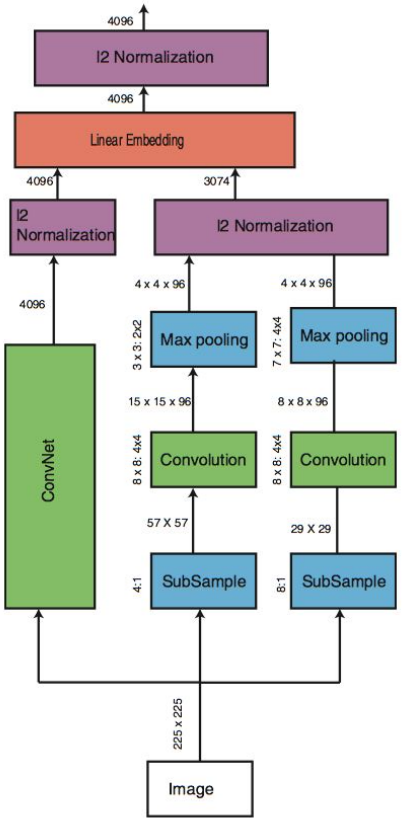
Notes - Software

Read a little bit more about using Keras CNNs

Followed some tutorials

Read a few papers about CNNS

Began constructing a basic sequential CNN using Keras in Python



12/01/2018

Goals

- Sequential model for CNNs
- Learn about CV
- Explore ways to detect breathing

Overall:

- Goals accomplished
- We could have spent more time learning about how CNNs work, but we began to ran out of time and set to building the basics of the model

CNN Model

```
class CNN:
    def __init__(self, dataFolder):
        self.data = dataFolder


    def constructModel(self):

        images = [] # X
        values = [] # Y
        for filename in os.listdir(self.data):
            img =
cv2.imread(os.path.join(self.data,filename)
)
            if img is not None:
                images.append(img)

values.append(int(filename.split("_")[1][:1])
)


        self.model = Sequential()
        X_train, X_test, y_train, y_test
=train_test_split(images, values, test_size
= 0.2, random_state = 42)
```

Goals

- **Sequential model for CNNs**
- **Learn about CV**
- **Explore ways to detect breathing**

Overall:

- **Goals accomplished**
- **We could have spent more time learning about how CNNs work, but we began to ran out of time and set to building the basics of the model**

12/01/2018

Notes - Hardware

We decided to brainstorm different ways to capture breathing data. After searching online, we found examples that included using air temperature, and we thought of our own such as using a stretch, noise, or gyro based system.

Air temperature was very well documented online, however it required a tube be placed within the nose, or very close to the nose, to record the temperature of the ambient air, and the warmer breath, using that to differentiate intake vs outake. We decided to not go with this approach as it required face mounting, and was rather intrusive, which would be awkward for the user to use over a long period of time.

Goals

- Sequential model for CNNs
- Learn about CV
- Explore ways to detect breathing

Overall:

- Goals accomplished
- We could have spent more time learning about how CNNs work, but we began to ran out of time and set to building the basics of the model

12/01/2018

Notes - Hardware (cont)

The next system stretch, was relatively straightforward, as we thought of just splitting the voltage, and then using a variable resistor, that changes in resistance by length, and then measure the amount of voltage not passing through the resistor. This implementation seemed simple at first, however we were not sure if it would be reliable/stable enough, and we measured the chest expansion to be ~2cm, so we were not sure if we could reliably capture it.

An interesting option we explored was actually attempting to listen to their breath, and then detecting if they are wheezing during an asthma attack, however we could not find any data sets online for wheezing that had enough data, and it would be difficult to make a data set, causing us to scrap this idea.

Goals

- Sequential model for CNNs
- Learn about CV
- Explore ways to detect breathing

Overall:

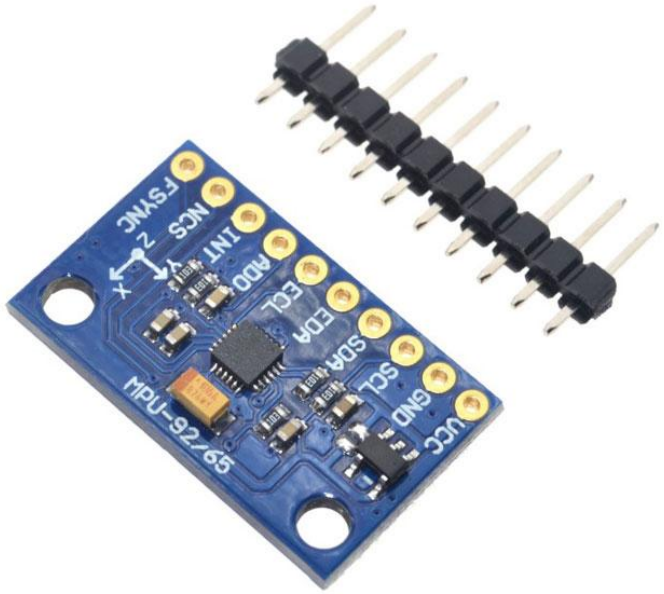
- Goals accomplished
- We could have spent more time learning about how CNNs work, but we began to ran out of time and set to building the basics of the model

12/01/2018

Notes - Hardware (cont)

Our final idea is a gyro based system, where we take multiple gyro sensors, and place them around the chest, and finding the change in distance between the sensors, to see the expansion of the chest.

Because it was distance calculations in a 3d space, and measuring it over time, we decided to go for this model as it was simpler.



12/06/2018

Goals

- Make helper functions for image saving and crafting Python
- Outlier detection methods on at least 3 tactics, compare
- Learn how a gyro sensor works

Overall:

- Goals accomplished ½
- Only made 1 method of the three desired, but it works ok. Will do the rest if time permits.

Notes - Software

Built a tool which will read all the files from a directory

For every image in the directory, it will:

- Read the image file name
- Read the image itself
- Append the image to an X array
- Take the image file name and read it's custom naming convention(number_YValue.png) to

```
def csvToImages(csv FileName, outputFolderName):
    if not (os.path.exists(os.path.join(os.getcwd(), outputFolderName))):
        os.mkdir(os.path.join(os.getcwd(), outputFolderName))


    # Read the csvFileName
    csv = pd.read_csv(csvFileName, delimiter=',')
    num = csv.shape[0]
    # For every filename
    for i in range (0, num):
        # First, get the filename
        currFile = csv.iloc[i, 0]
        # Get the csv file associated with the file
        df = dataToDf(currFile, " ")
        # Generate an image, name it i_0or1forAsthma and put it in
outputFolderName
        x = np.array([1,2,3,4,5,6,7,8,9,10, 11, 12])
        labels = pd.DataFrame(data = x)
        dataToImg(labels, df, outputFolderName + "/" + str(i) + "_" +
str(csv.iloc[i,1]))
```

The overall master csv file to image saving function, written utilizing all the other helper functions we have created so far.

```
def getOutliers(X):
    """
    # Detecting outliers based on the Modified Z-score method
    threshold = 3.5



    median = np.median(X)
    median_absolute_deviation = np.median([np.abs(x - median) for x in
X])
    modified_z_scores = [0.6745 * (x - median) /
median_absolute_deviation for x in X]
    return np.asarray(np.where(np.abs(modified_z_scores) > threshold))
    """
    # Detecting outliers based on the IQR Method
    quartile_1, quartile_3 = np.percentile(X, [25, 75])
    iqr = quartile_3 - quartile_1
    lower_bound = quartile_1 - (iqr * 1.5)
    upper_bound = quartile_3 + (iqr * 1.5)
    # print(X)
    # print("Lower: " + str(lower_bound) + ", Upper: " + str(upper_bound))
    lowers = np.asarray(X[X < lower_bound])
    uppers = np.asarray(X[X > upper_bound])



    # Prevents error when using np.concatenate
    if (lowers.size == 0):
        # print("The outliers are : " + str(uppers))
        return uppers
    if (uppers.size == 0):
        # print("The outliers are : " + str(lowers))
        return lowers



    toRet = np.concatenate((lowers, uppers))
    # print("The outliers are : " + str(toRet))
    return toRet
```

This code is a piece of code to detect outliers from an array,using the modified Z-score method, but converted into efficient, concise Python code

12/06/2018

Goals

- **Make helper functions for image saving and crafting Python**
- **Outlier detection methods on at least 3 tactics, compare**
- **Learn how a gyro sensor works**

Overall:

- **Goals accomplished ½**
- **Only made 1 method of the three desired, but it works ok. Will do the rest if time permits.**

Notes - Hardware

Tested with gyro sensor, show arduino code for a gyro sensor, had high error, and deviated slowly.

Gyro code:

```
#include "MPU9250.h"

// IMU readable object
MPU9250 IMU(Wire,0x68);
//enabled boolean
int status;

void setup() {
  // start serial to run
  Serial.begin(115200);

  //wait until serial is ready
  while(!Serial) {}

  // start imu
  status = IMU.begin();
}
```

12/06/2018

Goals

- Make helper functions for image saving and crafting Python
- Outlier detection methods on at least 3 tactics, compare
- Learn how a gyro sensor works

Overall:

- Goals accomplished ½
- Only made 1 method of the three desired, but it works ok. Will do the rest if time permits.

Notes - Hardware

```
void loop() {
  //update values
  IMU.readSensor();
  Serial.println("Acceleration: ");
  Serial.print("x: ")
  Serial.print(IMU.getAccelX_mss(),6);
  Serial.print("  y: ");
  Serial.print(IMU.getAccelY_mss(),6);
  Serial.print("  z: ");
  Serial.println(IMU.getAccelZ_mss(),6);

  Serial.println("Gyroscope: ");
  Serial.print("x: ");
  Serial.print(IMU.getGyroX_rads(),6);
  Serial.print(" y: ");
  Serial.print(IMU.getGyroY_rads(),6);
  Serial.print(" z: ");
  Serial.println(IMU.getGyroZ_rads(),6);
}
```

12/12/2018

Goals

- Research training data
- Further CNN model

Overall:

- I did the research
- The CNN model was scrapped
- We learnt about the pitfalls of CNNs, and now are migrating
- Overall, lost time but still somewhat successful.
- Found no data online
- Gyro was too unreliable

Notes - Software

Began generating fake training data by researching some average breathing rates for children through adults and creating training data (about 15 input files).

Read these and made plots and began testing the CNNs.

I realized that the CNN idea, although novel and quite cool, was inefficient as there were huge amounts of unused whitespace in the image that did not have plot points.

I then decided to change the entire model from a CNN to a classification problem given a set of data

12/12/2018

Goals

- Research training data
- Further CNN model

Overall:

- I did the research
- The CNN model was scrapped
- We learnt about the pitfalls of CNNs, and now are migrating
- Overall, lost time but still somewhat successful.
- Found no data online
- Gyro was too unreliable

Notes - Hardware

Tried to accurately measure the deviation, and find out if there was anyway to decrease the drift

Started out by clamping down the gyro sensor, but it was still deviating.

We switched over the analog mode to internal mode to get higher precision, however it did not decrease any of the slowly built up error.

We then tried using all the gyros we had purchased, and then taking the average of three, which was arguably even more unreliable as they would sometimes cancel eachother out, or add on to each other, creating even larger drift

With calibration it improved, how ever, for calibration it required that we hold it down in place, which would be impractical for the user to do.

We also tried removing small movement, however it was taking too long, and caused real movement to be suppressed

12/19/2018

Goals

- **Further the normal NN class**
- **Try atleast 5 to 8 different models, as they should not be too difficult to find and implement**

Overall:

- **Did not get to all the models**
- **Still got through a few and did further NN class, overall somewhat success**

Notes - Software

Created a neural network implementation which constructs either a multilayer perceptron or a decision tree classifier and trains the data.

Built a class called NN that uses sklearn and some of the previously built helper functions to do a series of functions:

- A function which sets the input directory to a file which contains a list of all the training files
- A function which goes iteratively through the list, viewing all the training files. Then, it will read the csv from the training file. Using the same custom naming convention specified earlier, it reads the y-value associated with the training value. Then, it sends the training set into an X-array. Similar as before with the image data, but now with vectorized data

12/19/2018

Goals

- Look into other ways to record chest movement

Overall:

- Ordered stretch band sensor

Notes - Hardware

We went back to the drawing board to figure out which method we would use instead of using a gyro sensor

We decided to switch over to using the stretch band sensor, and bought it from adafruit.

12/24/2018

Goals

- **Further the normal NN class**
- **Test the training data and validate its reliability and similarity to real life data and such**

Overall:

- **Did make significant work on the NN model class**
- **Did not get to test and evaluate the data**
- **Overall, good progress. Happy.**

Notes - Software

To the neural network implementation., I added:

- A function which sets an internal model variable to a new Multi-Layer Perceptron from sklearn, which is essentially a neural network with 3 hidden layers of size 14,10, and 6 by default. This neural network seems cool and works well in theory but due to the lack of training data, is less than ideal. With large amounts of data, this is a strong model
- A function which sets an internal model to a decision tree classifier, which constructs a new Decision Tree Classifier from sklearn. This is a model which uses decision tree learning to go from the vector inputted to the target model. It is called predictive learning, and it is better for small datasets.

12/24/2018

Goals

- **Further the normal NN class**
- **Test the training data and validate its reliability and similarity to real life data and such**

Overall:

- **Did make significant work on the NN model class**
- **Did not get to test and evaluate the data**
- **Overall, good progress. Happy.**

Notes - Software (cont)

To the neural network implementation., I also added:

- A function which makes predictions on the data given and the internal model. It simply predicts from the input vector then sets an internal predictions value to the predicted value.
- A function to evaluate predictions, using a confusion matrix and classification report. This is mainly to be used in training and not in real life application.
- Getter and setter methods for the model, training data, and predictions.

12/24/2018

Goals

- **Order any other components needed**

Overall:

- **Ordered 9v barrel adapter**

Notes - Hardware

Band is still shipping.

Ordered a 9v barrel adapter to power arduino externally, without having to take power via usb

12/29/2018

Goals

- Further the normal NN class
- Test data
- Research real-life data

Overall:

- Did not make significant work on the NN model class
- Tinkered with data
- Happy

Notes - Software

Upon some inspection of the data and predictions based on the training data, along with some reading into how asthma attacks work, I realized that they do not last for a very long time, so there was no need to take the whole set of data.

I realized that I need to use metrics to determine what the outliers in the data are, find where they occur. If there are no outliers we know that there was no asthma attack during the time interval where the data was recorded.

If there are outliers and significant outliers, then we know there was an asthma attack during the time interval where the data was recorded.

12/29/2018

Goals

- Further the normal NN class
- Test data
- Research real-life data

Overall:

- Did not make significant work on the NN model class
- Tinkered with data
- Happy

Notes - Software (cont)

Need to account for outliers

To do this, I implemented a find outliers method in the helper python file. There are three methods for doing this - IQR and Z-score (modified and normal).

I experimented with both Z-score methods but found them unsatisfactory for the data type we had.

Then, I tried the standard IQR method and it proved to be quiet useful and pinpoint the time when the asthma attack was ‘occuring’ in the training data.

12/29/2018

Goals

- **Finalize any more component purchases**

Overall:

- **Re evaluated the premises of the hardware**
- **Ordered resistors**

Notes - Hardware

After further looking into our design of the strap, we realized that since we are only getting around 2cm of circumference change in the chest, we need the length of stretch cord to be short enough, that 2cm becomes a large percentage of it.

We decided that we would instead use around 10 cm, however with the resistance of the stretch cord, we realized we could increase the range of recorded values by adding a resistor before in series, so we decided to order a pack of resistors online.

01/05/2019

Goals

- Normalization / scaling of data
- Test outliers vs normalization
- Make predictor

Overall:

- Finished normalization vs testing outliers
- Did not get to finish predictor
- Have bugs before I finished

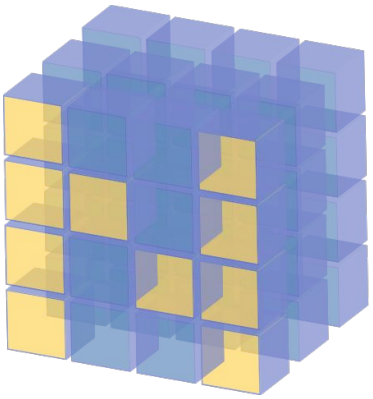
Notes - Software

Earlier, I had been trying to experiment with scaling the data, but this proved to be a significantly better method of altering the input data, so I removed the standard scaler.

I then had to change up the code significantly to account for this change.

I had to add many checks into the code so that when doing numpy functions on the data, I would not get errors.

This included some checks for empty arrays, checks for leading and trailing zeroes, and for things like that.



NumPy

01/05/2019

Goals

- Normalization / scaling of data
- Test outliers vs normalization
- Make predictor

Overall:

- Finished normalization vs testing outliers
- Did not get to finish predictor
- Have bugs before I finished

Notes - Software (cont)

Next, I integrated the getOutliers method into all my code.
When making predictions on data, the code will first identify the outliers in the data.
If there are none, there is not an attack.
However, if there are outliers, then it is sent to the machine learning model to determine if they are unwanted irregularities or indicate an actual asthma attack.

I also ended up doing a lot of testing with various print statements due to some numpy errors, which ended up taking me about 4 hours to fix. This was mainly due to the aforementioned errors and some problems with both prediction evaluation and prediction making with the internal models.

01/05/2019

Goals

- **Create a working prototype**

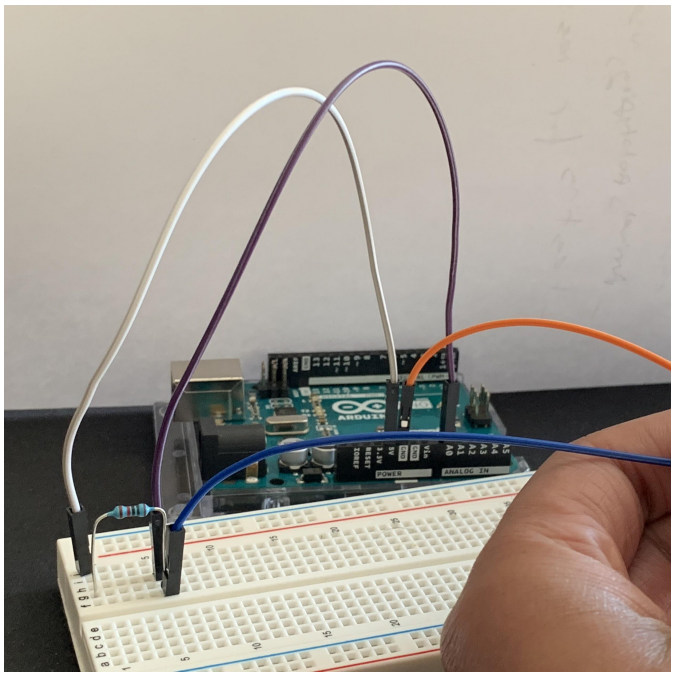
Overall:

- **Created a prototype, that could record values of the stretch band's current length**

Notes - Hardware

The adafruit band, and the resistors arrived, so we began working on our prototype

We used an arduino uno as our microcontroller, and from there used the 5v port to power a row on the breadboard.



01/05/2019

Goals

- **Create a working prototype**

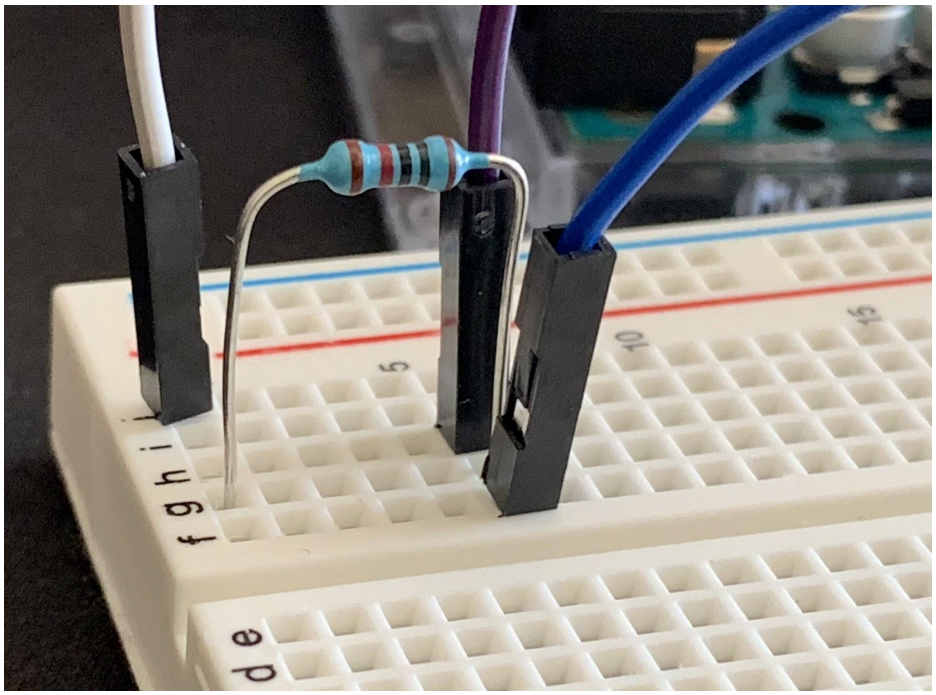
Overall:

- **Created a prototype, that could record values of the stretch band's current length**

Notes - Hardware

From there we connected one 10k resistor, and connected it onto a different row.

With the new row powered, we split the voltage, one going straight to the A0 analog pin on the arduino, and the other going through the stretch band and back to ground.



Due to resistance increasing when stretching the band, it would cause less voltage to go via the band, and more to go down to the A0 pin, allowing us to record the value.

01/05/2019

Goals

- **Create a working prototype**

Overall:

- **Created a prototype, that could record values of the stretch band's current length**

Notes - Hardware (Code)

```
#define pin A0

long count = 0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  double inputValue;
  count = count + 1;
  inputValue = analogRead(pin);
  Serial.print(count);
  Serial.print(",");
  Serial.println(inputValue);
  delay(10);
}
```

Started off by defining a millisecond counter as a global variable, along with the pin input. During setup, we switched on the serial to start printing values. During loop, we read the value, and then print it along with count.

01/12/2019

Goals

- Fix all the bugs
- Determine validity of data
- Do research on asthma

Overall:

- Fixed bugs
- Fixed outliers - not a direct goal, but important
- Tinkered with data a bit but not much
- No asthma research
- Overall, 75% success or so.

Notes - Software

I still had some bugs from the previous session that I had yet to fix.
My np.where function which I had utilized to write my get outliers code was oddly returning the index values of the outliers instead of the outliers themselves.
The index values are useful for determining the time of the attack, but are not useful for sending to the model, which requires the outlier values themselves.
To fix this, I needed to remove the np.where statement I used originally.
Instead, I broke up the IQR into two sections: lower bound outliers and upper bound outliers.
Then, to prevent errors when using np.concatenate to return the full list of outliers, I returned the nonzero array if one of the arrays was zero. If both arrays were 0, an empty array was returned. Otherwise, if both upper and lower outliers were found, they were concatenated and returned.

01/12/2019

Goals

- Fix all the bugs
- Determine validity of data
- Do research on asthma

Overall:

- Fixed bugs
- Fixed outliers - not a direct goal, but important
- Tinkered with data a bit but not much
- No asthma research
- Overall, 75% success or so.

Notes - Software (cont)

I also faced some issues with trimming the data and reshaping some of the errors. I repeatedly ran the code and identified errors, as numpy is quite finicky sometimes.

I slowly eliminated all the bugs through a trial and error method.

Lastly, I cleaned up the code by removing unnecessary commented code, removing any print statements, and adding a few comments as necessary.

01/12/2019

Goals

- Improve accuracy /precision of the

Overall:

- Found optimal resistance, and switched pin read to increase precision

Notes - Hardware

We realized that changing the resistor amount itself would affect the range of values, so we did testing to see if it would affect the standard deviation of values too:

Resistor	100k		10k		5k		100k, pos 2	
Average Per High Low	29.08	51.30693069	326.38	495.84	617.3	884	23.59	52.56
Range	22.22693069		169.46		266.7		28.97	
Average	40.205		411.11		750.65		38.075	
Deviation	1.461146637	4.277248121	1.721990861	3.883973806	1.872178011	3.946613431	1.211169143	4.168956946

01/12/2019

Goals

- **Improve accuracy /precision of the stretch cord**

Overall:

- **Found optimal resistance, and switched pin read to increase precision**

Notes - Hardware

As we increased the resistance, we increased the range of values, without significantly changing the standard deviation.

However as we got closer to the 1k resistor the upper values started to approach 1023, the maximum that the analog could take as input, so we stopped at 5k.

We also switched the pin input mode into

analogReference(INTERNAL);

As it would increase the precision of the readings, by switching 5v to 1.1v

01/19/2019

Goals

- Research NNs
- Research different classifiers
- Work on the classifiers if time

Overall:

- Got to research
- Did not get to work, but it's ok./

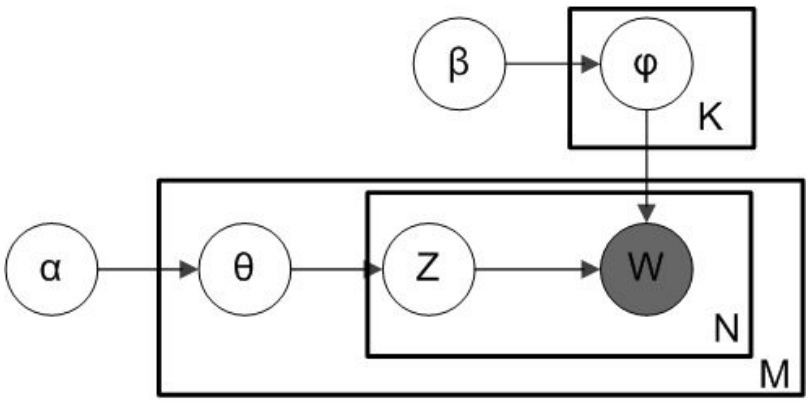
Notes - Software

From some research I conducted from outside sources, I confirmed that neural networks need huge amounts of data, as I previously had known.

However, apparently the decision tree classifier I had been previously relying upon was also a model which required lots of data.

I did some more research on various sklearn models and then learned that the SVM and Linear Discriminant Analysis models are extremely strong to use for small amounts of data.

LDA



01/19/2019

Goals

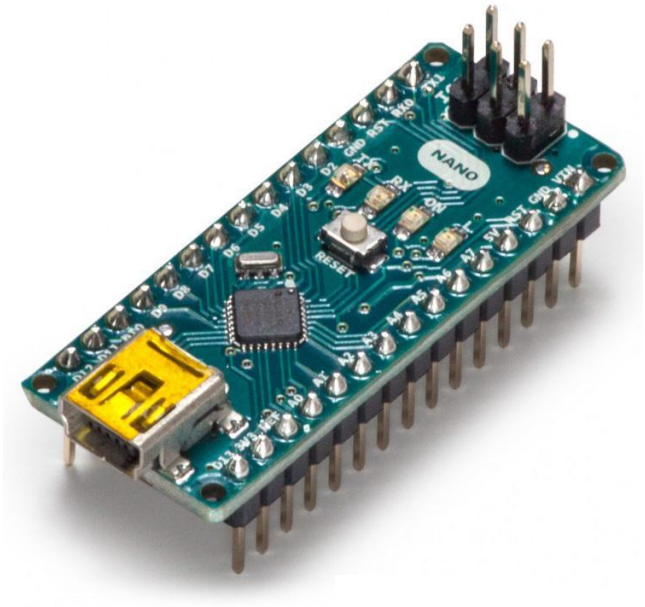
- Order components for final design

Overall:

- Ordered components

Notes - Hardware

Ordered arduino nano, and hc-05 to prepare for final version, and to connect to our application



01/24/2019

Goals

- Add LDA
- Add SVM
- Add and research other models

Overall:

- Tried LDA
- Added SVM
- Did not get to other models

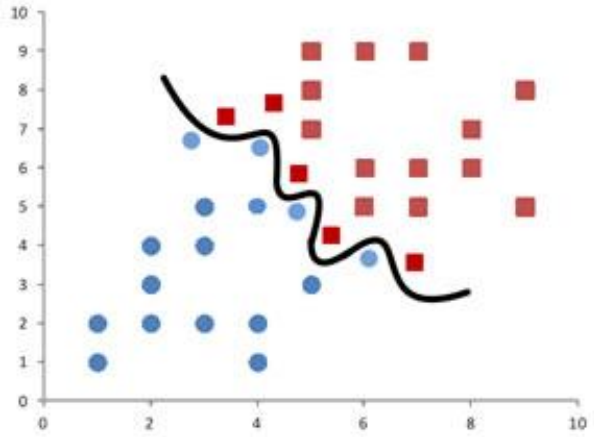
Notes - Software

I proceeded to add both of the LDA and SVM models into the code and tested them on the data samples I had artificially generated.

The LDA model did not work well, so I removed it from the code.

I then began to experiment with some different SVM kernels in order to find the optimum sklearn model for the data.

However, this led me to a chain of errors, slime easily solvable, some difficult.



01/24/2019

Goals

- Add LDA
- Add SVM
- Add and research other models

Overall:

- Tried LDA
- Added SVM
- Did not get to other models

Notes - Software(cont)

Numpy error which horribly stumped me.

- I got an error saying “ValueError: setting an array element with a sequence.”
- I eventually traced this error to be because the X_train had multiple arrays of different sizes.
- To fix this, I needed to pad each array in a manner such that the actual predictions and training were not affected. This produced another huge problem in itself. Solving this was a huge issue. I ran into many, many errors with using numpy pad algorithm. It did not work, repeatedly throwing the same ValueError exception. After a long time, I ended up essentially hardcoding this section. I made a new array, and did a for-each loop for the existing X_train array. Then, I copied each element in the existing X_train into the new array as a numpy array, and then called a custom pad function on it.

01/24/2019

Goals

- Add LDA
- Add SVM
- Add and research other models

Overall:

- Tried LDA
- Added SVM
- Did not get to other models

Notes - Software(cont)

- Upon some further testing, this achieved the desired effect.
- However, yet another error appeared! Now, when calling fit on the SVC, I got another ValueError. This time, it said: ValueError: X.shape[1] = 1 should be equal to 3, the number of features at training time.
- Luckily, the actual fitting (training) of the SVM was fixed. This error occurred during prediction time. I sat down to crack this error also.
- After a quick bit of testing, I managed to figure out that I needed to pad the prediction input X-array in a similar fashion in order to not upset the SVM.
- I changed around the NN class architecture to allow for some more self variables and getter methods. But surprisingly this didn't work either.

For now, a simple decision tree classifier would suffice.

01/24/2019

Goals

- Get bluetooth connection going to application with uno

Overall:

- HC-05 was incompatible
- Had to order new components

Notes - Hardware

After attempting to connect the HC-05 to the xcode application, it was not showing up, as it was bluetooth 2.0 which is incompatible with ios.

Ordered an HM-10, and a UART Slave only HC-08, as two alternative options we could try, that had fast cheap shipping on amazon.

01/29/2019

Goals

- **Finalize NN pipeline, add all the classifiers you want**

Overall:

- **Not much homework so I finished all my goals**

Notes - Software

I decided to play with a KNN also to see if it was easy to implement.

- If so, I planned on simply adding it into the code as an option for a classifier.
- It is not as optimized as the SVM for small data, but is still a formidable classifier.
- When building this classifier, the process was significantly smoother than when attempting to build the SVC.
- I ran into a few reshaping data errors that took no more than 15 minutes to solve. However, oddly, I got a valueError, although this valueError was simply due to an automatic wrapping of certain ints and data types when being parsed by Numpy algorithms. I changed the arrays to numpy arrays before performing the necessary transformations on them, then converted them to a list to copy over and use in the model.

```
class NN:
    def __init__(self):
        np.set_printoptions(threshold=np.nan)
        print("New Neural Network")

    def setInputDirectory(self, input):
        self.input = os.path.join(os.getcwd(), input)
        print("Set input directory to " + str(self.input))

    def collectData(self):
        print("Collecting data, saving")
        # Get all the inputs
        X = []
        Y = []
        csv = pd.read_csv(self.input)
        num = csv.shape[0]
        # For every input csv file in the input file
        for i in range (0, num):
            currFile = csv.iloc[i, 0]
            hasAsthma = csv.iloc[i,1]
            # Get the csv file associated with the file
            df = dataToDf(currFile, " ")
            x = df.iloc[0,:].values
            x = getOutliers(x)
            if x.size == 0: # Making sure no empty inputs for training
                x = np.asarray([])#.reshape(x.shape[1]))

            X.append(x)
            Y.append(hasAsthma)

        self.x = np.asarray(X)
        self.y = np.asarray(Y)
```

Neural network pipeline, current (part 1)

```
    def constructMLP(self, hidden = (14, 10, 6), iters = 1000):
        print("Constructing a Multi-Layer Perceptron")
        # CHANGE THE RATIO FOR TEST_SIZE TO SOMETHING IF TRAINING PLUS
OPTIMIZATION
        X_train, X_test, y_train, y_test =train_test_split(self.x, self.y, test_size = 0, random_state
= 42)
        self.y_test = y_test # Only saving the variable we need - y_test for evaluations
        mlp = MLPClassifier(hidden_layer_sizes = hidden, max_iter=iters)
        # 3 layers, one of 14 nodes, one of 10 nodes, one of 6 nodes
        mlp.fit(X_train, y_train)
        self.model = mlp

    def constructDTC(self):
        print("Constructing a Decision-Tree classifier")
        # CHANGE THE RATIO FOR TEST_SIZE TO SOMETHING IF TRAINING PLUS
OPTIMIZATION
        X_train, X_test, y_train, y_test =train_test_split(self.x, self.y, test_size = 0, random_state
= 42)
        self.y_test = y_test # Only saving the variable we need - y_test for evaluations
        DTC = DecisionTreeClassifier()

        DTC.fit(X_train, y_train)
        self.model = DTC

    def constructKNN(self):
        print("Constructing a K-Neighbors Classifier")
        # CHANGE THE RATIO FOR TEST_SIZE TO SOMETHING IF TRAINING PLUS
OPTIMIZATION
        X_train, X_test, y_train, y_test =train_test_split(self.x, self.y, test_size = 0, random_state
= 42)
        self.y_test = y_test
        classifier = KNeighborsClassifier(n_neighbors=5)

        X_train = X_train.reshape((-1,1))
        y_train = y_train.reshape((-1,1))

        print("X_train shape : " + str(X_train.reshape((-1,1)).shape))
        print("y_train shape : " + str(y_train.shape))

        classifier.fit(X_train, y_train)
        self.model = classifier
```

Neural network pipeline, current (part 2)

```
    def constructSVC(self, kernel = 'rbf', degree = 3): # Default value is the best kernel
        # Kernels = 'linear', 'poly', 'rbf','sigmoid'
        print("Constructing a Support Vector Machine")
        # CHANGE THE RATIO FOR TEST_SIZE TO SOMETHING IF TRAINING PLUS
OPTIMIZATION
        X_train, X_test, y_train, y_test =train_test_split(self.x, self.y, test_size = 0, random_state
= 42)
        self.y_test = y_test # Only saving the variable we need - y_test for evaluations

        max(X_train, key=lambda coll: len(coll))
        largest = max(X_train, key=len)
        print(largest.shape[0])

        self.largest = largest

        new_X_train = []
        for te in X_train:
            arr = np.asarray(te)
            if not arr.shape[0] == largest.shape[0]:
                print(arr)
                if (arr.shape[0] == 0):
                    arr = np.zeros((3))
                else:
                    arr = np.pad(arr, (0, largest - arr.shape[0]), 'constant', constant_values = (0,0))
            new_X_train.append(arr)

        if (kernel == 'poly'):
            SVM = SVC(kernel = 'poly', degree = degree)
        else:
            SVM = SVC(kernel = kernel)

        new_X_train = np.asarray(new_X_train)
        SVM.fit(new_X_train, y_train)
        self.model = SVM
```

Neural network pipeline, current (part 3)

```
def makePredictions(self, data):
        print("Predicting")
        data = getOutliers(data)
        np.trim_zeros(data)
        #print("The data post trimming is: " + str(data))

        """ Include bottom section if using SVC's cuz they're STUPID """
        #
        # if data.size == 0: # No outliers? No asthma attack
        #     self.predictions = 0
        # else:
        #     if (data.shape[0] == 0):
        #         data = np.zeros((3))
        #     else:
        #         data = np.pad(data, (0, self.largest.shape[0] - data.shape[0]), 'constant',
constant_values = (0,0))
        #
        #     data = np.reshape(data, (data.size, 1))
        #     predictionsnew = self.model.predict(data)
        #     self.predictions = predictionsnew

        """ Remove this bottom section if using SVC's cuz they're STUPID """
        if data.size == 0:
            self.predictions = 0
        else:
            data = np.reshape(data, (data.size, 1))
            predictionsnew = self.model.predict(data)
            self.predictions = predictionsnew

    # Only used for optimization of model
    def evaluatePredictions(self):
        print("Evaluating")
        print("Test: " + str(self.y_test))
        print("Predictions: " + str(self.predictions))
        print(confusion_matrix(self.y_test, self.predictions))
        print(classification_report(self.y_test, self.predictions))

    def getPredictions(self):
        return self.predictions

    def returnModel(self):
        return self.model
```

Neural network pipeline, current (part 4)

```
def main():
    nn = NN()
    nn.setInputDirectory("data\\allInputs.txt")
    nn.collectData()
    nn.constructKNN() # linear, poly, rbf, sigmoid
    testInput = np.array([6, 10, 13, 15, 18, 37, 21, 13, 14, 11, 9,
12]).reshape(1,-1)
    nn.makePredictions(testInput)
    print("Predictions for the data are: " + str(nn.getPredictions()))

if __name__ == "__main__":
    main()
```

Neural network pipeline, current (part 5)

01/29/2019

Goals

- **Established bluetooth connection**

Overall:

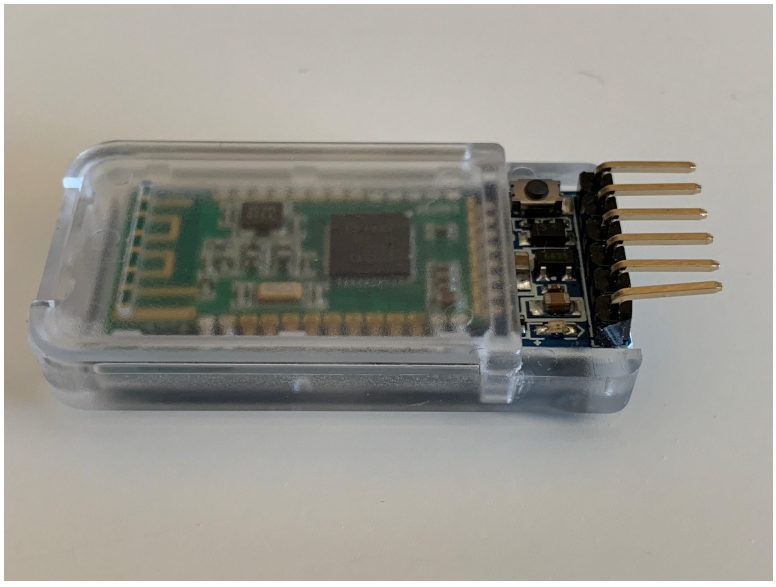
- **HC-08 received data, however it could not transmit to the phone**
- **HM-10 was able to transmit to the phone**

Notes - Hardware

HC-08 was promising at first, as it was successfully being scanned, connected, and was controlling the LED on the arduino uno reliably.

However after writing to the serial using the stretch sensor values, it was not transmitting to the phone. After attempting to fix it by explicitly defining the serial and using it, however it still was not working.

After switching it over to the HM-10 however, it started transmitting the values successfully



02/11/2019

Goals

- Implement new algorithms
- Test real data
- Make any adaptations

Overall:

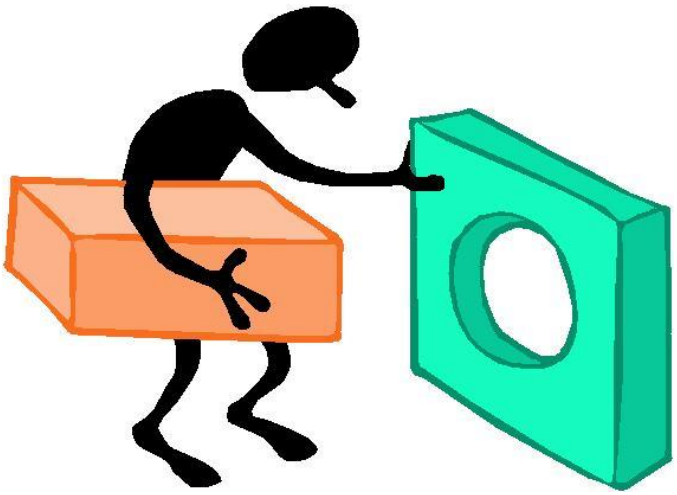
- Backwards progress but now we start from the right foot
- Got good data

Notes - Software

Up to this point, we had taken a short break to learn the algorithms available, their different requirements, and how to deal with short amounts of data.

Around this point, hardware began to put together the entire device and was able to get me some real training data.
I tested all the old models on the new code, but found that the old code could not adapt properly to the new data, which is the appropriate input which the final app will be receiving.

Incompatible



02/11/2019

Goals

- **Implement new algorithms**
- **Test real data**
- **Make any adaptations**

Overall:

- **Backwards progress but now we start from the right foot**
- **Got good data**

Notes - Software (cont)

I removed all the old code. Although this was a disappointing moment, we had learned so much from our initial code and endeavors, including
How to solve a variety of errors from

- Numpy Value Errors
- Import errors using matplotlib and other Tkagg dependent libraries
- Numpy Floating Point and DataType errors

How many models worked, and the mathematical reasoning behind

- KNN models
- LDA models
- SVM / SVC models
- Sequential CNN models
- General NN architecture

02/11/2019

Goals

- **Implement new algorithms**
- **Test real data**
- **Make any adaptations**

Overall:

- **Backwards progress but now we start from the right foot**
- **Got good data**

Notes - Software (cont)

Further, about Python, we learned:

- How to make classes about a neural network, and contain entire models and their helper functions in an object
- How to organize our code effectively
- How to troubleshoot code throughout the use of try / except statements and print statements.
- How to traceback an error effectively and identify the exact segment of the code statement that is causing the error

Overall, we were disappointed for a couple moments before we realized that although we had been set back in terms of time, we had been pushed forward significantly in terms of knowledge and experience, which are some of the most important things we had hoped to gain from this project.

02/11/2019

Goals

- **Transmitting stretch sensor values**

Overall:

- **Fixed baud rate to allow values to send**

Notes - Hardware (cont)

After transmitting values, we tried transmitting over the analog value, however at 10ms, it was not sending the data at all.

After debugging, we realized sending only the count worked, however adding the resistance values cause it to fail.

We measured how much data we were sending we realized that we sent data that required a higher bitrate than we had specified by the baud rate.

By switching the baud rate from 9600 over to 11520, it began working again.

02/17/2019

Goals

- Determine best placement of the device

Overall:

- Found best placement
- Also played with some new data and acquired best possible sparsity of data

Notes - General

- We used measuring tape to figure out how the human chest moves when breathing. We attempted to account for multiple factors, including
  - Muscle content (how would people with larger chests breathe)
  - Possible asthma attacks (simulating hyperventilating to get accurate measurements)
  - Possible vibrations or shuddering in

02/17/2019

Goals

- **Determine best placement of the device**

Overall:

- **Found best placement**
- **Also played with some new data and acquired best possible sparsity of data**

Notes - Software

Plotted out the new data to get a feel for how it works, approximation of a sinusoidal function, determined how the data moves.

Took some data and played with the graphing of it on multiple intervals - data collected every 10 ms, every 100s, every 50ms, etc, etc.

We found that data every 100ms was too weak to detect change, as it lacked the precision of the 10ms data

Decided to stick with the 10ms data, at least for the initial process

The data could be adapted and the models changed later if we found that we needed to optimize the data and model in order to use less space and be more power efficient.

02/17/2019

Goals

- **Solder the prototype**

Overall:

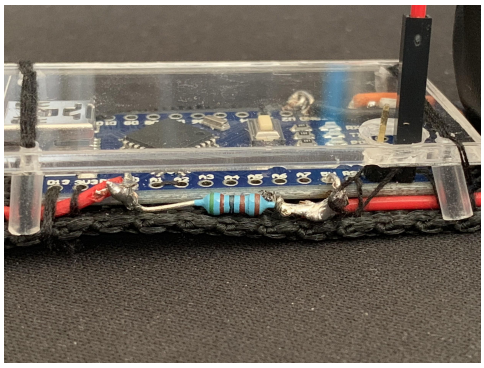
- **Ended up soldering the stretch band and nano, off of the band**

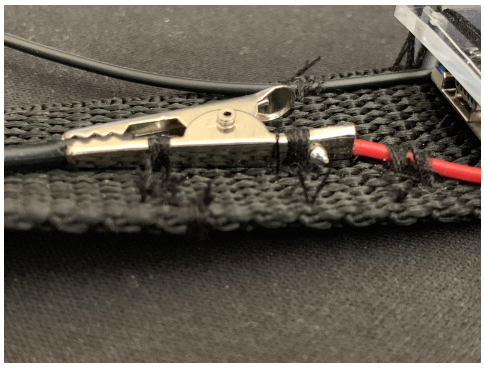
Notes - Hardware

We started off by taking the arduino nano and placing it on the band, and spacing out the battery and bluetooth placement.

We tried to stitch the resistor onto the band under the nano, however it was hard to solder it first before stitching, and stitching it first made it harder to solder.

We then decided to directly solder the resistor from the 5v straight to the ground removing the splitter for ground. From there we branched the stretch cord by soldering a cable off the resistor, and connecting it to an alligator clip holding one end of the stretch band.





We then took the stretch band's other end and connected it to gnd.

02/24/2019

Goals

- **Try and create a sinusoidal predictor for the data to act as a nice overlay for the user when they interact with the app**

Overall:

- **Tried to create a predictor, can’ say it worked. LassoCV should not be linear but it is here for some reason.**

Notes - Software

For our model to work, we need to do many things

- First, take txt file into csv file
- Plot csv file out on an elongated timeseries plane using pyplot for visualization
- Fit a polynomial model to the data to do predictions and get an approximation of how the breath data moves.
- Plot the polynomial approximations on top of the original data
- Use a certain model or code function in order to isolate every ‘breath’, or period of the sinusoidal function we create.

```
def oldModelCode():
    N = dfx.size
    t = np.linspace(0, 4*np.pi, N)
    f = 1.15247


    guess_mean = np.mean(dfy)
    guess_std = 3*np.std(dfy)/(2**0.5)/(2**0.5)
    guess_phase = 0
    guess_freq = 1
    guess_amp = 1


    data_first_guess = guess_std*np.sin(t+guess_phase) + guess_mean


    optimize_func = lambda x: x[0]*np.sin(x[1]*t+x[2]) + x[3] - dfy
    est_amp, est_freq, est_phase, est_mean = leastsq(optimize_func,
[guess_amp, guess_freq, guess_phase, guess_mean])[0]


    data_fit = est_amp*np.sin(est_freq*t+est_phase) + est_mean


    # recreate the fitted curve using the optimized parameters


    fine_t = np.arange(0,max(t),0.1)
    data_fit=est_amp*np.sin(est_freq*fine_t+est_phase)+est_mean


    plt.plot(t, dfy, '.')
    plt.plot(t, data_first_guess, label='first guess')
    plt.plot(fine_t, data_fit, label='after fitting')
    plt.legend()
    plt.show()
```

I tried to build a custom sinusoidal model using scipy functions but ran into many ValueError bugs, so I left that idea alone

```
lasso_eps = 0.0001
    lasso_nalpha=20
    lasso_iter=5000
    # Min and max degree of polynomials features to consider
    degree_min = 2
    degree_max = 8

    # Test/train split
    X_train, X_test, y_train, y_test = train_test_split(dx, dy,test_size=
0.05)
    # Make a pipeline model with polynomial transformation and LASSO
regression with cross-validation, run it for increasing degree of
polynomial (complexity of the model)

    model1 =
LassoCV(cv=10,verbose=0,normalize=True,eps=0.001,n_alphas=100,
tol=0.0001,max_iter=5000)
    model1.fit(X_train,y_train)
    y_pred1 = np.array(model1.predict(X_train))

    RMSE_1=np.sqrt(np.sum(np.square(y_pred1-y_train)))
    print("Root-mean-square error of Metamodel:",RMSE_1)



    py = model1.predict(dx)

    fig, ax = plt.subplots(figsize=(size))
    ax.scatter(dx, py, s = 15) # Thicker width for easy identification by
model
    ax.axis('off')
    fig.savefig(name + "pred_")
    plt.close(fig)
```

I moved on to creating a custom model using a LassoCV. This achieved extremely high error and needs to be fixed, but first I wished to address the problem of excess unnecessary data

02/24/2019

Goals

- **Solder the bluetooth**

Overall:

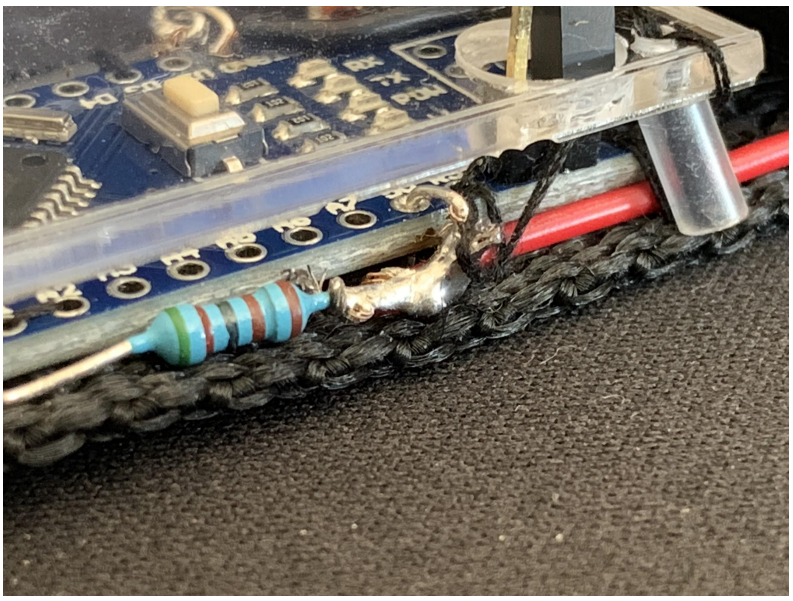
- **Tried to create a predictor, can’ say it worked. LassoCV should not be linear but it is here for some reason.**

Notes - Hardware

The bluetooth module needed 3.3v, so we used a digital output, and placed it into high mode. We then attached the ground and tested it, however it was not getting power.

After using a multimeter we realized it was outputting around 3v, which was underpowering the bluetooth module.

We utilized the regulator on the bluetooth module, so that we can directly input 5v, by branching off of the resistor that is on the stretch sensor.



02/24/2019

Goals

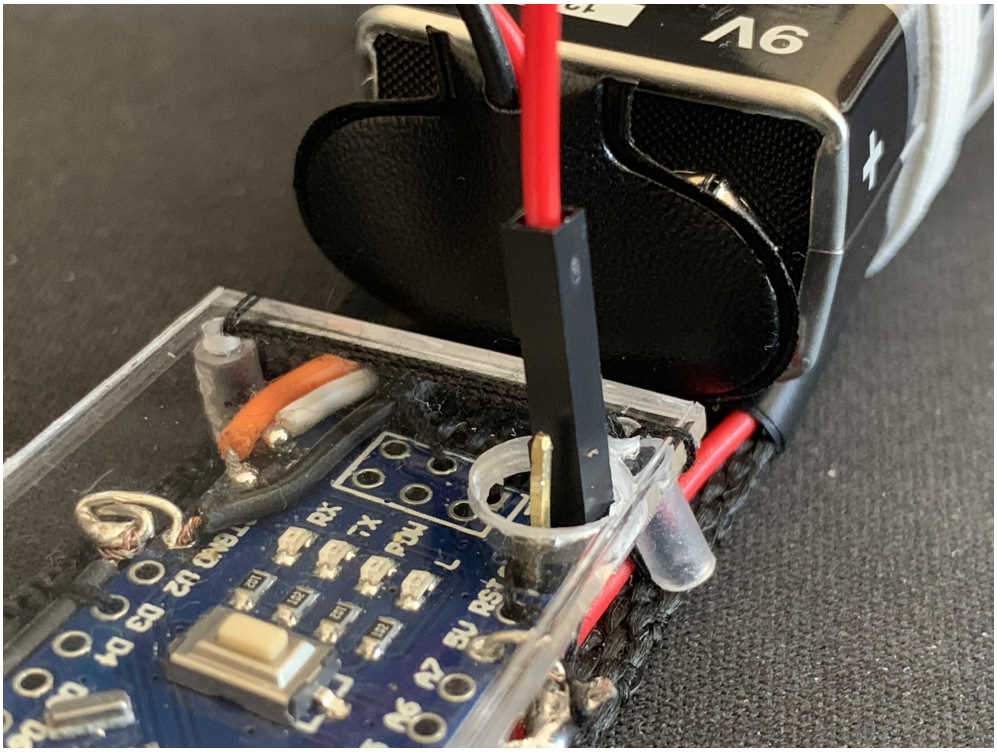
- Solder the bluetooth

Overall:

- Tried to create a predictor, can’ say it worked. LassoCV should not be linear but it is here for some reason.

Notes - Hardware

The battery clip was hard to attach, so instead of clipping and unclipping battery, we added pins to allow the adapter to be unplugged.



03/02/2019

Goals

- I want to create functions to clean the data

Overall:

- Only wrote 1 function to clean
- Instead, wrote slope functions
- Overall, success because of good progress

Notes - software

I created a function called getIrrelevantData that detected where the irrelevant data ends, and then the model would only train on the code which contains robust data.

This means that if a certain section of the data was to have a std of less than 1, meaning it is very static and nonmoving, then that section would be cut out - it simply negatively affects the sinusoidal model we create.

For some odd reason, the code does not take a subset of the data which should work, it instead makes the values which do not work null.

I needed to write a function to explicitly drop the na rows, which was not an issue and just took a simple two lines.

```
def getIrrelevantData(data):


    counter = 0
    while (data.size - counter) > 100:
        snapshot = data[counter:counter + 100]
        std = np.std(snapshot)
        if (std > 10):
            break


        # We only go up by 10 so we have a shifting window, most
accurate
        counter += 10


    return counter
```

Getting irrelevant data function

03/02/2019

Goals

- I want to create functions to clean the data

Overall:

- Only wrote 1 function to clean
- Instead, wrote slope functions
- Overall, success because of good progress

Notes - software (cont)

I made a normalization function, and then used some statistics to calculate when the data was going above and below a certain ‘standard deviation’.

I created a localization mean and standard deviation, and data points within two times that original localization and exceed the local standard deviation are said to be of a different group.

Doing this, I grouped all the data into sections. I calculated slopes based on the sections and used this to create the model.

03/02/2019

Goals

- I want to create functions to clean the data

Overall:

- Only wrote 1 function to clean
- Instead, wrote slope functions
- Overall, success because of good progress

Notes - software (cont)

Made a python function to calculate the slopes of the data.

Then, it takes the slopes and finds points of inflections, returning both the slopes of the data and the points of inflection of that data (where the slopes of the two subsequent data points are of different signs, indicating a change in slope, and hence a point of inflection)

This did not work because the data is extremely sparse, so I changed the slope calculator to instead take the average slope of every 4 points, just for testing and determining where the outliers and such lie.

The avg in 4 did not work, so I changed to avg on 6 and then skipping a few values ahead to make the slope calculations more sparse.

```
def calculateSlopes(data, skipFactorForCalculatingSlopes, size):
    """
    Go from 0 to data.size - 1
    Calculate the slopes between points i and i + 1
    Check where the slopes turn negative, put a big red dot there on plot
    """



    diffs = []
    rc = data["resistance"].values
    for i in range(0, data["ms"].size - size):
        diff = avgSlope(rc, i, i + size)
        i += skipFactorForCalculatingSlopes
        diffs.append(diff)


    poi = []


    for i in range(len(diffs) - 1):
        if (diffs[i] < 0.04 and diffs[i] > -0.04):
            poi.append(i)


    poi_scipy = np.asarray(argrelextrema(data["resistance"].values,
np.less))


    return {"slopes":diffs, "interest":np.asarray(poi)}
```

Calculate slopes function

03/02/2019

Goals

- Clean up the device

Overall:

- Stitched it on securely
- Redid the gap for stretch band
- Added cover

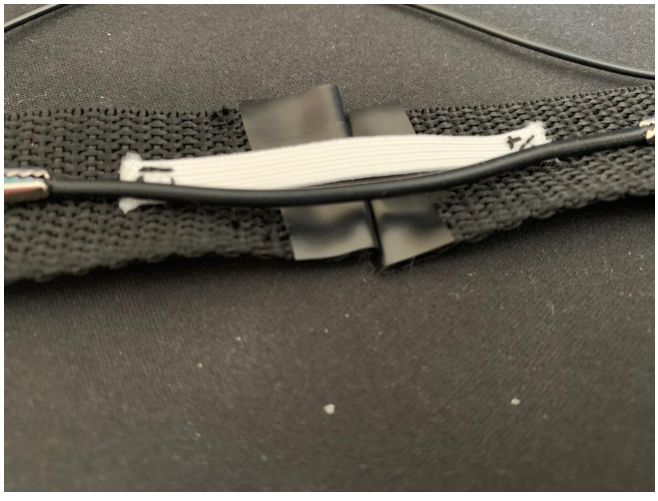
Notes - Hardware

We started off by stitching everything onto the band, and cut a section, to allow it to expand, and use the stretch sensor

The stretch sensor itself was not strong enough to close the gap back together, so we used a piece of elastic to drag them together.

We also added a plastic covering to the arduino nano, to protect the microcontroller.

We finally stitched down all the cables, to make sure they could not shake, and disconnect the solder creating faulty connections



03/09/2019

Goals

- **Change all the current code to work better with longs.**
- **Further cleaning process, cut down to minimums only in the data**

Overall:

- **Changed all code**
- **Fixed minor bugs**
- **Did a lot of work on the slope process**
- **Did not get to minimums only, but got very far**

Notes - Software

UNFORTUNATELY, we had accidentally been typecasting all the measurements to an int. All our problems would have been solved if we instead dealt with longs.

- Changed resistor bands to get much more robust values
- Tested all old algorithms and strategies on the new data
- Found that the averaging algorithm across 6 to 8 points served the best
- Rewrote averaging code to be more intuitive and optimized for 8 data points

03/09/2019

Goals

- **Change all the current code to work better with longs.**
- **Further cleaning process, cut down to minimums only in the data**

Overall:

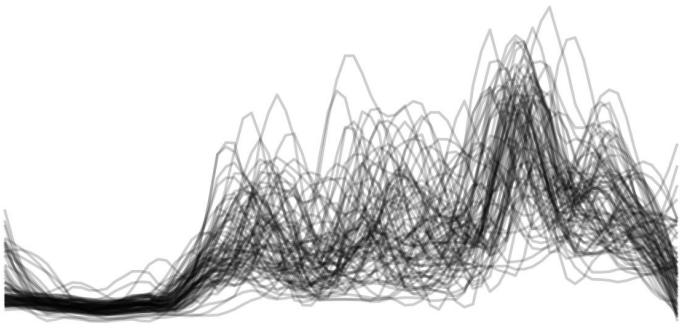
- **Changed all code**
- **Fixed minor bugs**
- **Did a lot of work on the slope process**
- **Did not get to minimums only, but got very far**

Notes - Software

The point of inflection detector is picking up many unnecessary values near the beginning of the graph. We plan to

- Make a copy of the data and normalize it
- Check that the std is past a certain threshold before letting that data go on

This way, we remove the unnecessary data near the beginning which is a lot of empty, straight data.



03/09/2019

Goals

- Change all the current code to work better with longs.
- Further cleaning process, cut down to minimums only in the data

Overall:

- Changed all code
- Fixed minor bugs
- Did a lot of work on the slope process
- Did not get to minimums only, but got very far

Notes - Software

Now, the data is plotting at an odd offset, where it seems to be shifted to the left.

After some tinkering, I found that we need to shift the entire dataset, when showing it to the screen, over to the right I found that the constant we needed to move the x by was the initial counter used for determining where the useful code starts, called beginningOfUsefulData

```
def removeNullValues(data, beginning):
    data["resistance"] = data["resistance"][beginning:]
    data["ms"] = data["ms"][beginning:]
    # removing null values
    data = data[pd.notnull(data['ms'])]


    return data


def removeSimilarValues(data, beginning):
    for i in range(beginning, len(data["ms"]) - 1):
        if (data["ms"][i] == data["ms"][i + 1]):
            data["ms"][i] = np.NaN

    Return data
```

More cleaning code

- *Removing null values from data*
- *Removing similar subsequent values from data*

03/09/2019

Goals

- Change all the current code to work better with longs.
- Further cleaning process, cut down to minimums only in the data

Overall:

- Changed all code
- Fixed minor bugs
- Did a lot of work on the slope process
- Did not get to minimums only, but got very far

Notes - Software

Now, we have another issue, although this one, in theory, is easier to solve.

- For each data point, we have to determine if it is a local maximum or minimum. This is fairly simple and can be done by comparing it to the location of the next and previous point of interest.
  - Less than prev, less than next = minimum
  - More than prev, more than next =

03/09/2019

Goals

- **Change all the current code to work better with longs.**
- **Further cleaning process, cut down to minimums only in the data**

Overall:

- **Changed all code**
- **Fixed minor bugs**
- **Did a lot of work on the slope process**
- **Did not get to minimums only, but got very far**

Notes - Software

As I was looking through the graphs generated by my current code, I found that the code will occasionally not find a data point, maximum or minimum, due to extreme sparsity of the data around that point.

- To counter this, I changed the groupings of the slopes to 4, in order to calculate more slopes and hence find more point of interests.
- This did not work, and as I was examining the graphs, I noticed that the density of the data points was extremely high near the minimums.
- Since we had captured approximately realistic data, I decided to try a higher grouping to see how the slope calculator would adapt, and if it would recognize the minimums as a point of interest now.

03/09/2019

Goals

- **Change all the current code to work better with longs.**
- **Further cleaning process, cut down to minimums only in the data**

Overall:

- **Changed all code**
- **Fixed minor bugs**
- **Did a lot of work on the slope process**
- **Did not get to minimums only, but got very far**

Notes - Software

Simultaneously, I developed a function to remove data points which were next to each other and the same value. What this does is remove unnecessary noise, and then help in removing noise from our slope function so it is more robust based on the area of breath that is analyzing, thus less susceptible to being skewed heavily by noise from random measurements.

- What this does is set those similar / same values to null
- When this is done, and then a numpy function is used to detect and remove NaN numbers, the entire data is shifted significantly over. This means that the previous methods, which placed the points of interest based on a certain offset, needed to be rewritten entirely.

03/09/2019

Goals

- **Change all the current code to work better with longs.**
- **Further cleaning process, cut down to minimums only in the data**

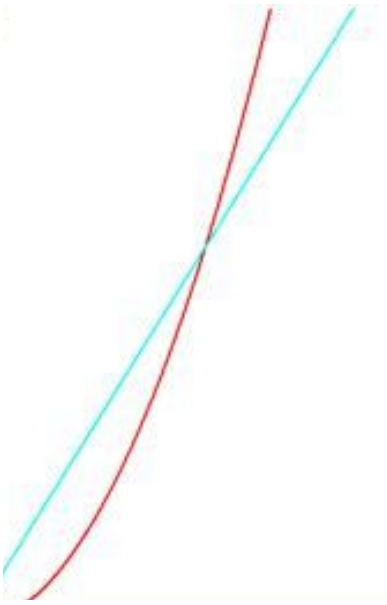
Overall:

- **Changed all code**
- **Fixed minor bugs**
- **Did a lot of work on the slope process**
- **Did not get to minimums only, but got very far**

Notes - Software

I rewrote the necessary methods and then tested based on a ‘slope range’ of 3. When viewing the output data, I noticed

- Some of the minimum points are being ignored
- Some of the maximum points are being ignored
- The slopes of the ignored points did not seem to differ much, if at all, from the slopes of the recognized points of interest.



03/09/2019

Goals

- Change all the current code to work better with longs.
- Further cleaning process, cut down to minimums only in the data

Overall:

- Changed all code
- Fixed minor bugs
- Did a lot of work on the slope process
- Did not get to minimums only, but got very far

Notes - Software

I decided to adapt my slope approach. I changed it so that if the slope was close to 0 (absolute value of slope is < 0.03), then we would make it a point of interest. I wanted to see how this approach worked and then adapt it as needed, or discard the approach entirely if it was not powerful enough.

I noticed that a much larger number of the maximums and minimums were being noticed by the code now.

However, the concentration of the points of interests was extremely high at some areas.

In order to combat this, we would need to make a function which would go through the points of interest and cuts down very close clusters of points to 1 or two points.

I wrote this initial code, but it was not actually removing data points for some reason.

```
def removeClusters(data):
        remove = []
            for i in range(len(data) - 1):
                if (data[i + 1] - data[i] < 10):
                    remove.append(i)



        data = [x for x in data if x not in remove]
            return data
```

Oddly, for some reason, python shorthand was not working, so I had to use the more brute-force loops to remove all duplicate / close values.

```
def removeClusters(data):
    remove = []
    for i in range(len(data) - 1):
        if (data[i + 1] - data[i] < 10):
            remove.append(data[i])


    for x in remove:
        data.remove(x)
    return data
```

03/09/2019

Goals

- **Re-do bluetooth code and connection**

Overall:

- **Got bluetooth read write properly working**

Notes - XCode App

We had a rudimentary bluetooth connection working from earlier, which we decided to redesign

We used corebluetooth, and tried to scan for each device, but it was not reliable. We switched over to using the specific service id we set to the HM-10

When peripheral was discovered we would discover their services.

When services found scan for characteristics

Bluetooth characteristics scanning was not working at first, because we used the characteristic id in one section where we were supposed to put service, so rather than scanning the service for characteristics it was scanning characteristics for characteristics.

```
func startScan() {
    centralManager.scanForPeripherals(withServices: [CBUUID(string: "FFE0")], options:
        [CBCentralManagerScanOptionAllowDuplicatesKey : false])
}
```

03/09/2019

Goals

- **Re-do bluetooth code and connection**

Overall:

- **Got bluetooth read write properly working**

Notes - XCode App

After fixing that it gave the read option, so it was reading the data being broadcasted by the arduino nano, however the values were completely random. After logging the arduino output type, and xcode bluetooth input, we realized we were using the wrong integer encoder for reading the data

We reused a good amount of code from the old bluetooth communication, allowing it to get up and running smoothly.

One weird issue we had was when running it outputted 0 if above a certain value, however that error disappeared after cleaning the xcode project.

03/10/2019

Goals

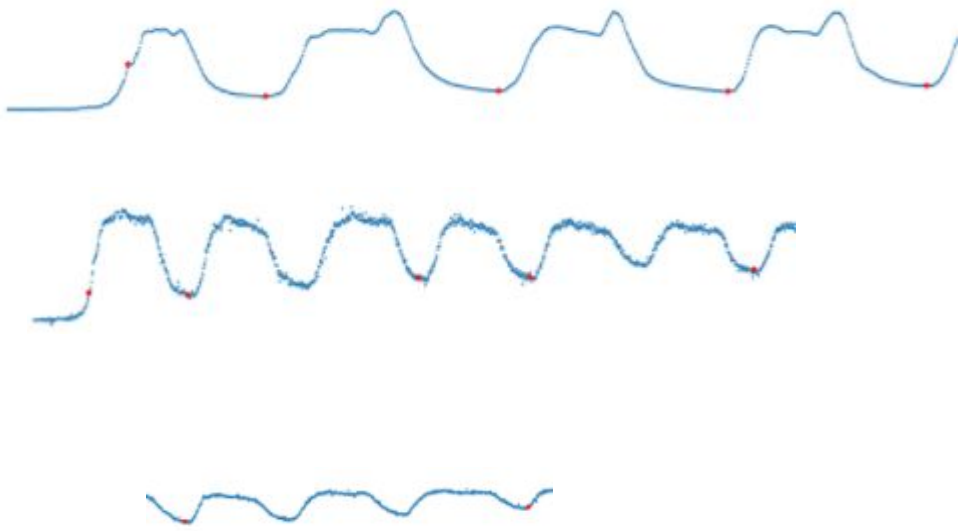
- Get max/min fix, optimize slope calculation and poi identification
- Clean up the python files, maybe centralize

Overall:

- I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.
- The python files are centralized & the pipeline is built. Happy.

Notes - Software

Picking up where I left off the previous day, I took the duplicate code and ran it on the data I had. This returned much better values for the approximate max and min. Now, the only problem was to get rid of fake minimum / maximums.



03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

To do this, I created a function called removeLocalMaxMins. This function is supposed to:

- Calculate the 'height' of every point of interest
- Calculate the approximate amplitude of the entire sinusoidal function by subtracting the smallest height from the average of all the heights of the points of interest in order to receive an approximate amplitude. This amplitude would be halved, and values which are next to each other with a difference less than the approximate amplitude would be removed.
- I ran into different errors while testing this, due to out of index values and also due to calculating the averages and such on the indexes instead of the actual values.

```
interesting_data = []
    for x in interests:
        interesting_data.append(data[x])


    # Typecast to numpy array
    interesting_data = np.array(interesting_data)


    # calculate the mean, then the minimum value, make a range
    avg_interest = np.mean(interesting_data)
    approx_amplitude = avg_interest - interesting_data.min()


    print("the approximate amplitude is " + str(approx_amplitude))


    remove = []


    for i in range(len(interesting_data) - 1):
        if (interesting_data[i+1] - interesting_data[i] < approx_amplitude/2):
            remove.append(copy[i]) # Append the INDEX to remove


    for x in remove:
        interests.remove(x)


    return interests
```

Code for removeLocalMaxMins, isolating only the minimums and maximums in the data

03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

This code removes ALL the maximums and unnecessary data.

This way, only the minimums are preserved. The minimums are used to identify where a new 'breath' is started.

The only issue now was, the code would cut out one really useful breath which was due to a outlier that could happen in the real testing of the project. So, the code had to be adapted:

- The amplitude was changed from over 2 to over 4, then the absolute difference between points was taken and if it was below that amplitude, the point was removed.
- This did not work either, so the amplitude was cut down to over 2.5, and it was able to differentiate fine.

03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

Now, we have to make a function to remove the tops, or maximums, from the data. For this, I made a function called removetopValues, which takes in the data and the interesting points. It calculates an amplitude of the data. Then, for selecting what points to remove:

- If the next point is lower than the current point, so the current point is a maximum AND the absolute value of next point minus the current point is past a certain threshold, then we can consider those two points to be contrasting max and mins. We delete the value at i, which is the maximum
- We also need to add a special case to address the last point in case it is a maximum, because then the code will not consider it in the original loop due to it being out of bounds.

```
def removeTopValues(data, interests):
    interesting_data = []

    for x in interests:
        interesting_data.append(data["resistance"][x])

    interesting_data = np.array(interesting_data)
    avg_interest = np.mean(interesting_data)
    approx_amplitude = avg_interest - min(interesting_data)

    remove = []

    for i in range(len(interesting_data) - 1):
        diff = interesting_data[i+1] - interesting_data[i]
        if (diff < 0 and abs(diff) > approx_amplitude / 2):
            remove.append(interests[i]) # Append the INDEX to remove

    if (interesting_data[len(interesting_data) - 1] > avg_interest):
        remove.append(interests[len(interesting_data) - 1])

    for x in remove:
        interests.remove(x)

    return interests
```

Removing the top values from the data as to isolate solely the minimums

Now, we need to create a function which will split the data into segments based on the minimums - in between each minimum is a segment. This function will also skip the beginning where the subject is not breathing.

```
def splitDataIntoSegments(data, interests):
    segments = []
    for i in range(0, len(interests) - 1):
        # segment is from current interest at i to next point at i + 1
        segment_1 = data[interests[i]:interests[i+1]]
        segments.append(segment_1)


    return segments
```

Now, finally, I needed a piece of code to plot and save the segments into a folder. This one gave me many errors due to global variables and weird shapes, so I had to rewrite a lot of method definitions.

```
def plotAndSaveSegments(segments, outputFolder, shapeDesired):
    counter = 0
    for segment in segments:
        counter += 1
        print(segment["ms"])
        print(segment["resistance"])
        dataToImg(segment["ms"], segment["resistance"], outputFolder +
"/" + "segment" + str(counter), [], shapeDesired)
```

03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

So far, we have been writing everything in a python file and testing them as random methods. However, we need to have a certain organization and encapsulation of the entire project so that it can be localized.

- We put the different types of functions in different files
- We put the proper imports
- We created a pipeline that:
- Reads, cleans, processes data
- Determines slopes, points of interests
- Makes all the folders necessary
- Creates a general plot, then each segment specific plot, and puts them in individual folders. Again, the entire thing is organized locally.

We had a lot of silly bugs with the pipeline, concerning global vs passed variables, parameter lists, etc. Finally, we finished.

03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

- Then, once the single data piece pipeline was done, I built the full_data piece, which simply requires a list of data files before it does the full pipeline. This worked well, and I added some more folder changes to organize the output images better.
- I tried to use a custom scipy function called find_peaks, which went horribly. After I edited a lot of my code to match that function and work with it, I re-tried my functions. Almost none of them worked, and I wasted another hour switching all my code back to using the lists that the original code utilized.
- I wanted to reset all my code and then restart from there and see what's going wrong currently.

03/10/2019

Goals

- **Get max/min fix, optimize slope calculation and poi identification**
- **Clean up the python files, maybe centralize**

Overall:

- **I got max/min fixed, got slope optimized, and poi identification, although slightly variant on bad data, is strong.**
- **The python files are centralized & the pipeline is built. Happy.**

Notes - Software

- Then, once the single data piece pipeline was done, I built the full_data piece, which simply requires a list of data files before it does the full pipeline. This worked well, and I added some more folder changes to organize the output images better.
- I tried to use a custom scipy function called find_peaks, which went horribly. After I edited a lot of my code to match that function and work with it, I re-tried my functions. Almost none of them worked, and I wasted another hour switching all my code back to using the lists that the original code utilized.
- I wanted to reset all my code and then restart from there and see what's going wrong currently.
- We tested new data which is more representative of an actual asthma attack. Our code did not fare well on that data.

I tweaked the code to have some more expandability and customizability, and then reran the model. Now, it fared decently, with some excess points in the beginning but otherwise good.
Custom average slope function and adapations:

```
def slope(x1, x2, ydiff):
    return (x1 - x2) / ydiff


def avgSlope(rc, beg, end):
    diff = 0
    for i in range (beg, end):
        diff += slope(rc[i], rc[i+1], 10)

    return diff / (end - beg)


def calculateSlopes(data, skipFactorForCalculatingSlopes, size):
    diffs = []
    rc = data["resistance"].values
    for i in range(0, data["ms"].size - size):
        diff = avgSlope(rc, i, i + size)
        i += skipFactorForCalculatingSlopes
        diffs.append(diff)

    poi = []

    for i in range(len(diffs) - 1):
        if (diffs[i] < 0.04 and diffs[i] > -0.04):
            poi.append(i)

    return {"slopes":diffs, "interest":np.asarray(poi)}
```

```
def pipeline(parameters):
    csvFileName = parameters["csv"]
    outputFolderName = parameters["output"]
    graphShapeGeneral = parameters["graph_shape"]
    skipFactorForCalculatingSlopes = parameters["skip"]
    testDataIter = parameters["iter"]
    size = parameters["size"]

    # Read the csvFileName
    data = dataToDf(csvFileName, delimiter = ",")

    # Create general output folder
    createOutputFolder(outputFolderName)

    # Get the useful data starting point
    beginningOfUsefulData = getIrrelevantData(data["resistance"])

    # Initial cleaning of the data
    removeSimilarValues(data, beginningOfUsefulData)
    removeNullValues(data, beginningOfUsefulData)

    # Edit interesting points list through top down approach
    interest = calculateSlopes(data, skipFactorForCalculatingSlopes, size)["interest"]
    interest = removeClusters(interest)
    interest = removeLocalMaxMin(data, interest)["pois"]
    interest = removeTopValues(data, interest)["pois"]

    # Plot the overall data
    dataToImg(data[["ms"]].values, data[["resistance"]].values, outputFolderName + "\\general",
"overall_iter_" + str(testDataIter), interest, graphShapeGeneral)

    # Get the segments and plot them
    segments = splitDataIntoSegments(data, interest)
    plotAndSaveSegments(segments, outputFolderName + "\\segments", (10, 5), testDataIter)


    print(anomalyDetector(segments))
```

The full pipeline for one piece of data

03/10/2019

Goals

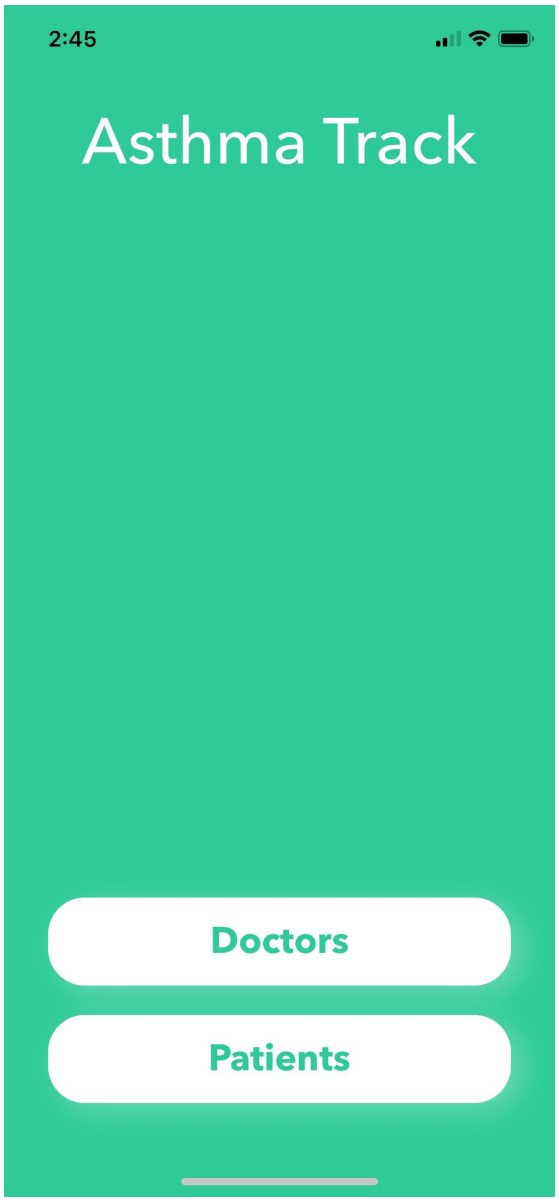
- Get a basic ui done, and get firebase connection working

Overall:

- Finished a ui, registration, login, and uploading breath data for python to receive

Notes - XCode App

Tried a darker theme at first but it was not very appealing, so we switched over to a lighter green.



03/10/2019

Goals

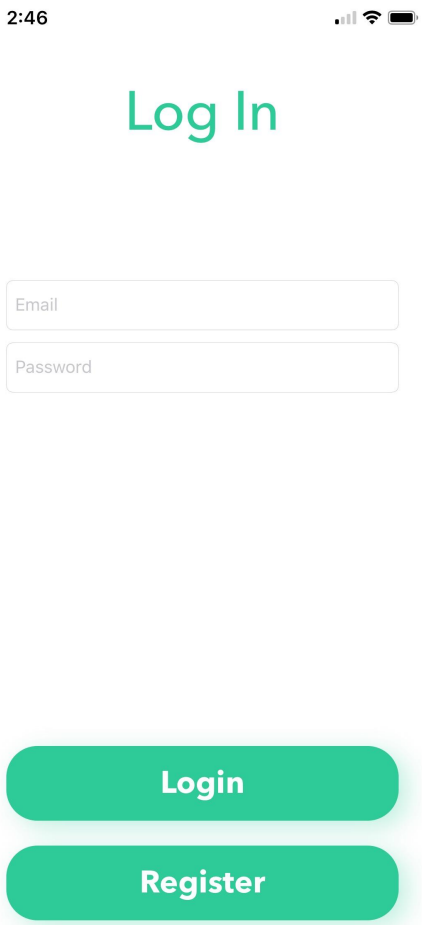
- Get a basic ui done, and get firebase connection working

Overall:

- Finished a ui, registration, login, and uploading breath data for python to receive

Notes - XCode App

Added register function through firebase however it was not working, and we were unsure as to why. After debugging it stated that we did not allow email sign in, as when we enabled it earlier we did not save and update the rules. Log in worked smoothly



03/10/2019

Goals

- Get a basic ui done, and get firebase connection working

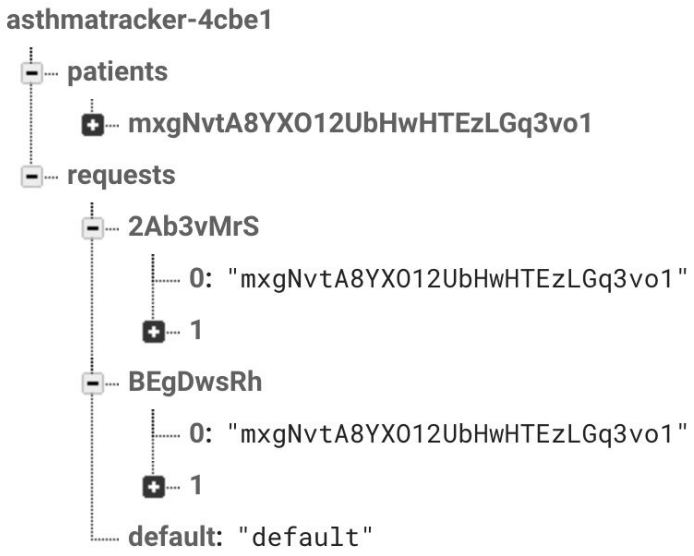
Overall:

- Finished a ui, registration, login, and uploading breath data for python to receive

Notes - XCode App

We decided to send data to firebase in packets of 10 seconds, under a request folder. It had a unique hash, that the python program could download, run through, and then send back to the user to be displayed later on through charts.

When we pushed data, there was an extra /r/n, which we had not noticed before, as it would become a new line when printing.



03/10/2019

Goals

- **Get a basic ui done, and get firebase connection working**

Overall:

- **Finished a ui, registration, login, and uploading breath data for python to receive**

Notes - XCode App

We debugged it for a while, until we realized it was not on the xcode side, so checked on the arduino side, and realized we used a println, when writing out the data, so after switching over to print, it worked.

03/11/2019

Goals

- **Test the algorithms on new data, variant data, identify strengths and weaknesses. Optimize the model.**
- **Fix all algorithms that are variant**

Overall:

- **Not much fixing of algorithms was needed**
- **The model was fairly optimized**
- **Strengths and weaknesses somewhat assessed.**

Notes - Software

Hardware was able to get me more data

We ran tests on different polling lengths and found the polling lengths that would work the best for the algorithms we had wrote. This way, the algorithms would be optimized with minimal changes.

I wrote an algorithm which would take the interest points and find an average distance between individual points.
The interest points which were very close to each other, within a third of that average, were said to be unnecessarily clustered, and thus removed.
This eliminated the main problem with our current data.

```
def removeVeryCloseInterestingPoints(interest):
    diffs = 0
    interest = interest.tolist()


    for i in range(len(interest) - 1):
        diffs = diffs + interest[i+1][0] - interest[i][0]


    avg = diffs / ( len(interest) - 1)
    remove = []


    for i in range(len(interest) - 1):
        if interest[i+1][0] - interest[i][0] < avg / 3:
            remove.append(interest[i])


    for x in remove:
        interest.remove(x)


    interest = np.asarray(interest)


    return interest
```

Removing very close points in the list of interesting points

03/11/2019

Goals

- **Test the algorithms on new data, variant data, identify strengths and weaknesses. Optimize the model.**
- **Fix all algorithms that are variant**

Overall:

- **Not much fixing of algorithms was needed**
- **The model was fairly optimized**
- **Strengths and weaknesses somewhat assessed.**

Notes - Software

Now, we need to write out anomaly detector.

The way this code works is it will first find out the first, max, and last value of an array.
Then, it will find their indexes.

If the distance from the min to the max is much smaller than the distance from the max to the end, it means that the person spent a long time breathing out the air they inhaled quickly - a characteristic sign of hyperventilation / asthma attacks.
So, that would mean we return true.

Otherwise, we return false.

```
def detectAnomaly(segment):
    rc = segment["resistance"]
    rc = rc.tolist()

    minValue = rc.min
    maxValue = rc.max
    # get min of segment
    minIndex = rc.indexOf(minValue)
    maxIndex = rc.indexOf(maxValue)
    totalDiff = segment["ms"].shape[0]


    if (maxIndex - minIndex) <= totalDiff * 2/5:
        return True

    return False
```

Current anomaly detection

```
def detectAnomaly(segment):
    rc = segment["resistance"]

    minValue = rc.min()
    maxValue = rc.max()
    # get min of segment
    rc = rc.tolist()
    minIndex = rc.index(minValue)
    maxIndex = rc.index(maxValue)
    totalDiff = segment["ms"].shape[0]


    if (maxIndex - minIndex) <= totalDiff * 2/5:
        return True


    return False
```

Now, I had hardware send me over 5 realistic data sets of about 5000 points, collected over the span of about 5 to 10 seconds. As I was mass-running the code, I ran into a few anomaly detection issues. I promptly resolved these issues by changing the anomaly code to:

```
def anomalyDetector(segments):
    anomaly = []
    for segment in segments:
        anomaly.append(detectAnomaly(segment))


    l = 0 # longest bad streak
    x = 0 # Current counter
    for i in range(len(anomaly) - 1):
        if (anomaly[i] == anomaly[i +1] && anomaly[i+1] == True):
            x += 1
        else:
            if (x > l):
                l = x
        if l > 2:
            return True
    if x > 2:
        return True

    return False
```

Similarly, in order to get 1 condensed value for the anomaly detector given a set of segments, I did:

03/11/2019

Goals

- **Test the algorithms on new data, variant data, identify strengths and weaknesses. Optimize the model.**
- **Fix all algorithms that are variant**

Overall:

- **Not much fixing of algorithms was needed**
- **The model was fairly optimized**
- **Strengths and weaknesses somewhat assessed.**

Notes - Software

Looking at the tens of charts that have been generated, the thousands upon thousands of data points, and the results of my various algorithms, I made a few findings.

- On sparse, long data, setting a low skip and a low size works very well
- On short, normal-ish data, setting a higher skip and a higher size works fine
- In general, higher skip and higher size increases speed. However, lower skip and low size tend to increase accuracy. Since we cannot predict the type of data we are being inputted, we must try to find a happy medium and settle with it.
- For now, even though it is slow, I will settle with skip of 2 and size for slope batch of 2

03/12/2019

Goals

- Finish poster board
- Finalize App
- Finalize engineering notebook

Overall:

- We worked on the poster board
- Got pretty far on ntbk
- Ran into bugs, didn't finish app

Not done, but good progress

Notes

We worked on the posterboard

We also worked on getting the database to communicate with the computer-based server in order to send the data and graphs generated by the programs over to the database.

03/12/2019

Goals

- **Finish poster board**
- **Finalize App**
- **Finalize engineering notebook**

Overall:

- **We worked on the poster board**
- **Got pretty far on ntbk**
- **Ran into bugs, didn't finish app**

Not done, but good progress

Notes - Software

We need to make a Pyrebase app that will:

- Connect to the master database
- Download the data from there every x seconds
- Parse the data, save it locally, then perform operations on it
- Take those operations and resulting anomalies and other interesting points and send it to the user. Send segments also to the database.
- The code for the initial, rudimentary Pyrebase app for downloading and saving is in the following pages.

```
import qrcode
import pyrebase

import os
import time

import pandas as pd
import numpy as np

from generalUtil import createOutputFolder

""" configuration """
config = {
  "apiKey": "AIzaSyBA2HdYTxzeXvIJedXm2Cx2MKpPv_2-pB0",
  "authDomain": "com.cvenkatramani.AsthmaAid",
  "databaseURL": "https://asthmatracker-4cbe1.firebaseio.com",
  "storageBucket": "asthmatracker-4cbe1.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()

""" loop itself"""

results = dict(db.child("requests").get().val())
print(results.keys())
keys = results.keys()

numKeys = 0
waitTime = 0
```

This is the imports and configuration for the Pyrebase app

```
while(True):
    for key in keys:
        if not key=='default':
            amps = [] # y
            mss = [] # x
            numKeys += 1
            for elem in (results[key][1]):
                curr = elem[:-2]
                ms = curr[:curr.index(",")]
                amp = curr[curr.index(",") + 1:]
                amps.append(amp)
                mss.append(ms)
            # Convert those lists into numpy arrays
            amps = np.array(amps)
            mss = np.array(mss)
            # concatenate them into a dataframe
            df = pd.DataFrame({"ms":mss, "resistance":amps})
            df.set_index("ms", inplace=True, drop=True)
            # save the dataframe as a csv
            createOutputFolder("inputsFromDatabase")
            df.to_csv("inputsFromDatabase/" + str(numKeys))
            db.child("requests").child(key).remove()
    time.sleep(waitTime)
```

This is the while loop itself for the Pyrebase app.

03/12/2019

Goals

- **Finish poster board**
- **Finalize App**
- **Finalize engineering notebook**

Overall:

- **We worked on the poster board**
- **Got pretty far on ntbk**
- **Ran into bugs, didn't finish app**

Not done, but good progress

Notes - Software

In order to run the full pipeline on the data we got, and then send the resulting values and such back to the database, we would need to edit the full Pipeline to return the values we need.

I edited the pipeline accordingly, returning a dictionary.

Then, I went over to the database file to start the edits I would need to do to run the pipeline on the data and then get the values to the database.

I needed to learn more about Pyrebase. After some tinkering and coding, I had a decent download, run functions, update loop going.

It will be detailed in the next two pages of code. No captions because of limited space.

```
import qrcode
import pyrebase

import os
import time

import pandas as pd
import numpy as np

from generalUtil import createOutputFolder
from singleDataPipeline import pipeline

""" configuration """
config = {
  "apiKey": "AIzaSyBA2HdYTxzeXvIJedXm2Cx2MKpPv_2-pB0",
  "authDomain": "com.cvenkatramani.AsthmaAid",
  "databaseURL": "https://asthmatracker-4cbe1.firebaseio.com",
  "storageBucket": "asthmatracker-4cbe1.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()

""" loop itself"""

results = dict(db.child("requests").get().val())
print(results.keys())
keys = results.keys()

# How far we've gone
numKeys = 0
# How long we wait between loops
waitTime = 10
# How long the program takes to run on avg, in seconds
offsetTime = 0
# Our "exit" condition - currently continuous
sentinel = True
```

```
while(sentinel):
    for key in keys:
        if not key=='default':
            amps = [] # y
            mss = [] # x
            numKeys += 1
            uid = results[key[0]]
            for elem in (results[key][1]):
                curr = elem[:-2]
                ms = curr[:curr.index(",")]
                amp = curr[curr.index(",") + 1:]
                amps.append(amp)
                mss.append(ms)
            # Convert those lists into numpy arrays
            amps = np.array(amps)
            mss = np.array(mss)
            # concatenate them into a dataframe
            df = pd.DataFrame({"ms":mss, "resistance":amps})
            df.set_index("ms", inplace=True, drop=True)
            # save the dataframe as a csv
            createOutputFolder("inputsFromDatabase")
            df.to_csv("inputsFromDatabase/" + str(numKeys))

            # Removing the current data to save space in the database
            db.child("requests").child(key).remove()

            # Run pipeline on the data
            parameters = {"csv":"inputsFromDatabase/" + str(numKeys) + ".csv", "output":
"output/graphs_" + str(i + 1), "graph_shape":(200,10), "skip":2, "iter":numKeys, "size":2}
            output = pipeline(parameters)

            # Getting the user wanted data and pushing it to them
            total_breaths = len(output["segments"])
            is_attack = output["overall_anomaly"]
            db.child("patient").child(uid).set({"is_attack":is_attack, "total_breaths":total_breaths})

            # Saving the segmentation data
            for i in range(len(output["all_anomalies"])):
                start = output["segments"][i][0]
                curr_data = {"start":start, "anomaly":output["all_anomalies"][i]}
                db.child("patient").child(uid).set(curr_data)

    # Remove this line later, just for testing to limit downloading
    sentinel = False
```

03/12/2019

Goals

- Finish poster board
- Finalize App
- Finalize engineering notebook

Overall:

- We worked on the poster board
- Got pretty far on ntbk
- Ran into bugs, didn't finish app

Not done, but good progress

Notes - Software

I am trying to run the software code to test the above pipeline for the database code, but I am getting a perplexing keyerror.

After some tinkering and some backtracking, I found that for some reason, data["resistance"] does not work.

The reason behind this is extremely perplexing, but I will do some more backtracing and debugging and find out for certain.

It took me a while to identify and trace back the source of this error, but I found that once the pipeline reads the csv file for the data inputted, it changes the columnn names to 0 and 1 instead of "ms" and "resistance" like they should be.

To fix this, I set the columns to "ms" and "resistance", and accordingly dropped the 0th row. It took me a bit to learn how to use df.drop properly, but I fixed the bug eventually.

03/12/2019

Goals

- **Finish poster board**
- **Finalize App**
- **Finalize engineering notebook**

Overall:

- **We worked on the poster board**
- **Got pretty far on ntbk**
- **Ran into bugs, didn't finish app**

Not done, but good progress

Notes - Software

Now, I have a bug where it says:

- TypeError: could not convert string to float: '509\r\r\n308,509'

Due to some random symbols being added when the data is initially pushed to firebase from the app. I tried to accommodate for this initially, but it is not working still.

I could not figure out why it is doing this, but implemented a barrier such that if the current element is more than 2 longer than the previous element, just pass it.

This raised another random error, :
ValueError: substring not found
Which I now need to research and find the answer to.

After a long chain of small errors here and there, I ran the code on the new data.
It fared decently, but needed to be updated to reflect the different data. I made some of the cuting proceses more lenient, and the interesting points fared better.

```
import qrcode
import pyrebase

import os
import time

import pandas as pd
import numpy as np

from generalUtil import createOutputFolder
from singleDataPipeline import pipeline

""" configuration """
config = {
  "apiKey": "AIzaSyBA2HdYTxzeXvIJedXm2Cx2MKpPv_2-pB0",
  "authDomain": "com.cvenkatramani.AsthmaAid",
  "databaseURL": "https://asthmatracker-4cbe1.firebaseio.com",
  "storageBucket": "asthmatracker-4cbe1.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()

""" loop itself"""
# db.child("medicine").child(Hash).set(data)

# Check for when the database changes

# Pyrebase to get a child listener
# From the child, from the 1, download all the data
# For each data point, remove the \r\n then we gucci

# Ignore default
vvvv = db.child("requests").get().val()
results = dict(vvvv)
keys = results.keys()
```

```
# How far we've gone
numKeys = 0
# How long we wait between loops
waitTime = 10
# How long the program takes to run on avg, in seconds
offsetTime = 0
# Our "exit" condition - currently continuous
sentinel = True

while(sentinel):
    for key in keys:
        if not key=='default':
            amps = [] # y
            mss = [] # x
            numKeys += 1
            uid = results[key][0]
            last = results[key][1][0]
            first_ms = results[key][1][0]
            first_ms = float(first_ms[:first_ms.index(",")])
            for elem in (results[key][1]):
                if len(elem) > len(last) + 2:
                    continue

                last = elem

                ms = float(elem[:elem.index(",")]) - first_ms
                amp = float(elem[elem.index(",") + 1:])
                #
                # try:
                #
                # except:
                #     continue

                amps.append(amp)
                mss.append(ms)
            # Convert those lists into numpy arrays
            amps = np.array(amps)
            mss = np.array(mss)
            # concatenate them into a dataframe
            df = pd.DataFrame({"ms":mss, "resistance":amps})
            df.set_index("ms", inplace=True, drop=True)
```

```
# save the dataframe as a csv
        createOutputFolder("inputsFromDatabase")
        df.to_csv("inputsFromDatabase/" + str(numKeys) + ".csv")

        # Removing the current data to save space in the database
        # db.child("requests").child(key).remove()

        # Run pipeline on the data
        name = "inputsFromDatabase/" + str(numKeys)
        parameters = {"csv": name + ".csv", "output": "output/graphs_" + str(numKeys),
"graph_shape":(200,10), "skip":2, "iter":numKeys, "size":2}
        output = pipeline(parameters)

        # Getting the user wanted data and pushing it to them
        total_breaths = len(output["segments"])
        is_attack = output["overall_anomaly"]
        print({"is_attack":is_attack, "total_breaths":total_breaths})
        db.child("patient").child(uid).set({"is_attack":is_attack, "total_breaths":total_breaths})

        # Saving the segmentation data
        for i in range(len(output["all_anomalies"])):
            start = output["segments"][i]
            curr_data = {"start":start["ms"].iloc[0], "anomaly":output["all_anomalies"][i]}
            print("Starts at " + str(curr_data["start"]) + ", the anomalies are : " +
str(curr_data["anomaly"]))
            db.child("patient").child(uid).set(curr_data)

    # Remove this line later, just for testing to limit downloading
    sentinel = False

    # Sleep approximately enough to wait for phone
    # to receive data, then repeat process until user cancels
    # time.sleep(waitTime - offsetTime)
```

This is still in the while loop, and it is the finishing bit of code.

This is the final working code.

03/13/2019

Goals

- **Finalize project**
- **Ensure full pipeline, hardware to software, works**
- **Finish abstracts**
- **Finish board**
- **Finish judges presentation**

Overall:

-

Notes - Software

Since pushing data to the database overwrites the current data, I had to rewrite the pipeline to ensure that it sent all the data at once, which would later be deleted once the app viewed the data.

I created arrays and appended values.

Then, I pushed all 4 arrays, for anomalies, breaths, attacks, and segments, to the database.

However, I was updated that I need to push everything based on hash values so that it is useable by the App itself.

I updated the code accordingly, so that it appended everything, including any current user information, to a master list, separated by hashes for each ‘data segment’, then uploads the entire thing in one swoop to the reader.

Arduino / Bluetooth

Python

- *Numpy, Pandas, Matplotlib, Sklearn*
- *Pyrebase / Firebase / Databases*
- *OOP / code organization*
- *How to troubleshoot code*
- *How to traceback an error*
- *CNN models*
- *CV Libraries, Keras, etc*
- *Classification models (KNN, SVC, LDA)*
- *NN models, architectures*

iOS

- Bluetooth / Arduino Connections
- Authentication / Databases

Essential skills

- *Value of hard work*
- *Persistence*
- *Commitment*
- *Debugging and Patience*