Cody Antonio Gagnon
3-11-20
TECHIN513 – Managing Data Signal Processing
Homework 3 – DSP, Feature Engineering and Dataset Generation

Bed Occupancy Detection & Recognition Using Force Sensitive Resistors (FSR)

I am using an ESP32 and its two onboard ADCs to detect values from an FSR, or Force Sensitive Resistor. These are sensors which decrease in resistance (Ω) as pressure is applied to them. The particular FSR I'm using is the SF15-600, an approximately 24", rectangular FSR, which is rated for up to 10kg.

Walfront SF15-600
amzn.to/3aoAkWs

Pressure can be applied anywhere on the sensor, including multiple places, and it will register as a change in voltage to an ADC. The decrease in resistance leads to an increase in voltage (V) in the circuit. Thus, we can measure and model the change in voltage as the change in force, where a higher voltage is equal to higher force on the sensor. It's important to note that the specific ESP32 board I'm using (DOIT ESP32 DEVKIT V1 Board) has a 12-bit SAR ADC w/values between 0 - 4095 and a sampling frequency of 6 kHz, but still struggles with low (0-0.1 V) and high (3.2-3.3 V) voltages due to the ADC itself. In this case we still get a lot of useful data, but as a result, some values could be missing (most of the time our values are over 0.1 V but there are some instances where they just register as 0 or 0.1 V). It's also important to note that while on its own, we can measure the voltage of this circuit in Volts (V), adding a secondary, constant resistor, allows us to calculate the FSR's resistance in Ohms (Ω) using the voltage divider equation. The layout of my hardware components in this sensor systems are as follows:

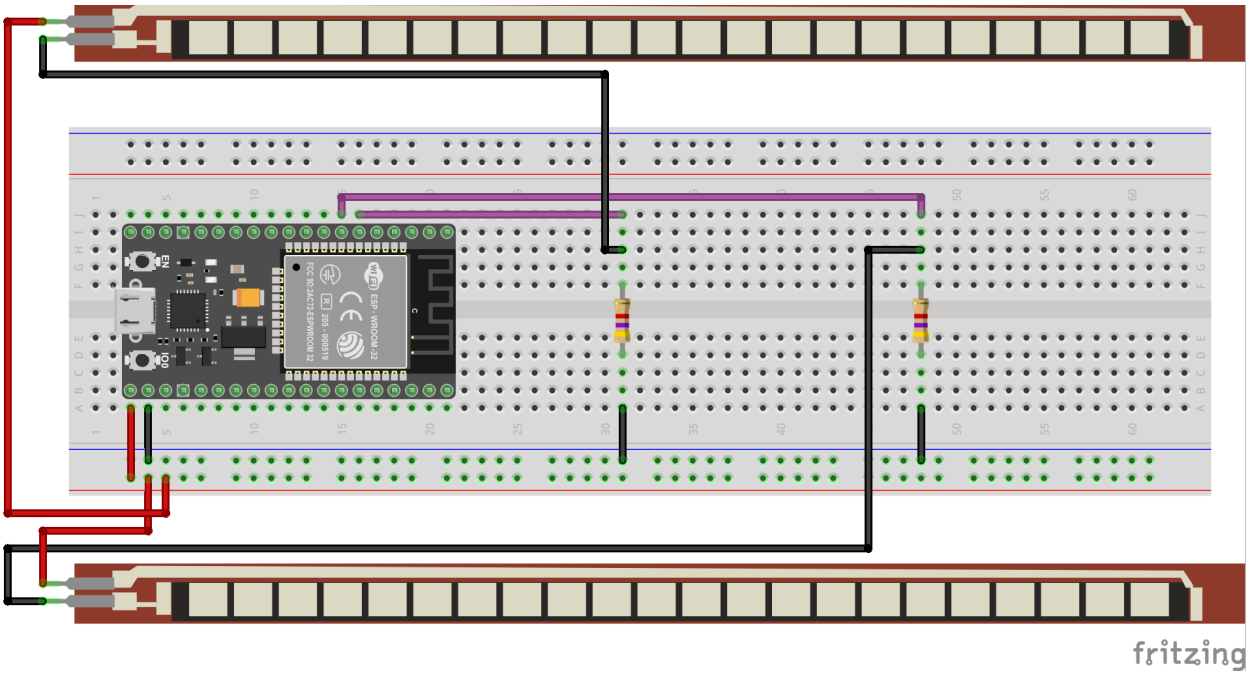


Figure 1: ESP32 & Sensors Hardware Configuration Diagram

This is a very simple circuit in which we have two Force Sensitive Resistors powered by the 5V line and two 4.7 kΩ resistors setup and connected to ground. The purple lines each represent a respective connection to a GPIO pin.

On the software side, one can use the function analogRead(PIN) on an Arduino (where PIN is an ADC GPIO pin), to get a voltage measurement from the FSR. After testing this works, we move to an amazing open source project called ESPHome, which allows you to flash ESP devices over the air after initial configuration (WiFi updates) and use templates to determine how to use the I/O of your device. There are many useful templates here which led me to the following configuration:

Figure 2: ESP32 Config Using ESPHome

In this YAML file we can see there are two ADC sensors defined (one for each side of the bed). Going through the configuration, I want to point out that attenuation is just a value that needed to be added due to the ADC of the ESP32. The update_interval means we get a reading every second.

We are using these ADC sensors as input to resistance sensors as well, which are doing the voltage divider equation for us to output the resistance in ohms (Ω) of our circuit. An important note here for the software setup is that the GPIO pins are located closer to the ground than the positive end, which changes the voltage divider circuit a bit. In this case it's a simple configuration option which you'll see above is set to UPSTREAM. When I measured the resistance of each resistor it was closer to 4.6 kΩ, so I chose that as the constant resistor value (even though I have 4.7 kΩ resistors in my circuit).

I am pushing the voltage data from this sensor into Home Assistant where it will be processed for automations. It is simple to set a threshold for the voltage detected by the ADC based on the FSR and use that to determine whether or not the bed is occupied. This can also be done on the ESP32 itself and output a simple discrete value for based on a threshold.You can also create a sliding window in ESPHome which can run on-device and do some minor processing of the data

to smooth things out and get an average value. My full workflow for using the data in this system and performing DSP and ML on it is as follows:

Figure 3: Overview of Technologies Used

1. ESPHome is used to get data into 2. Home Assistant. A good time-series database is needed to store and later query the readings which are taking place every second, which is where 3. influxdb comes in. I also downloaded the .csv files for Jupyter directly from the influxdb web interface. To help graph these data for visual analysis I use 4. Grafana. To explore this data and parse it in Python, I use 5. Jupyter notebooks (specifically Google Colab). Using Machine Learning in 6. Orange we can see if we're able to recognize which user is on a sensor given previous data.

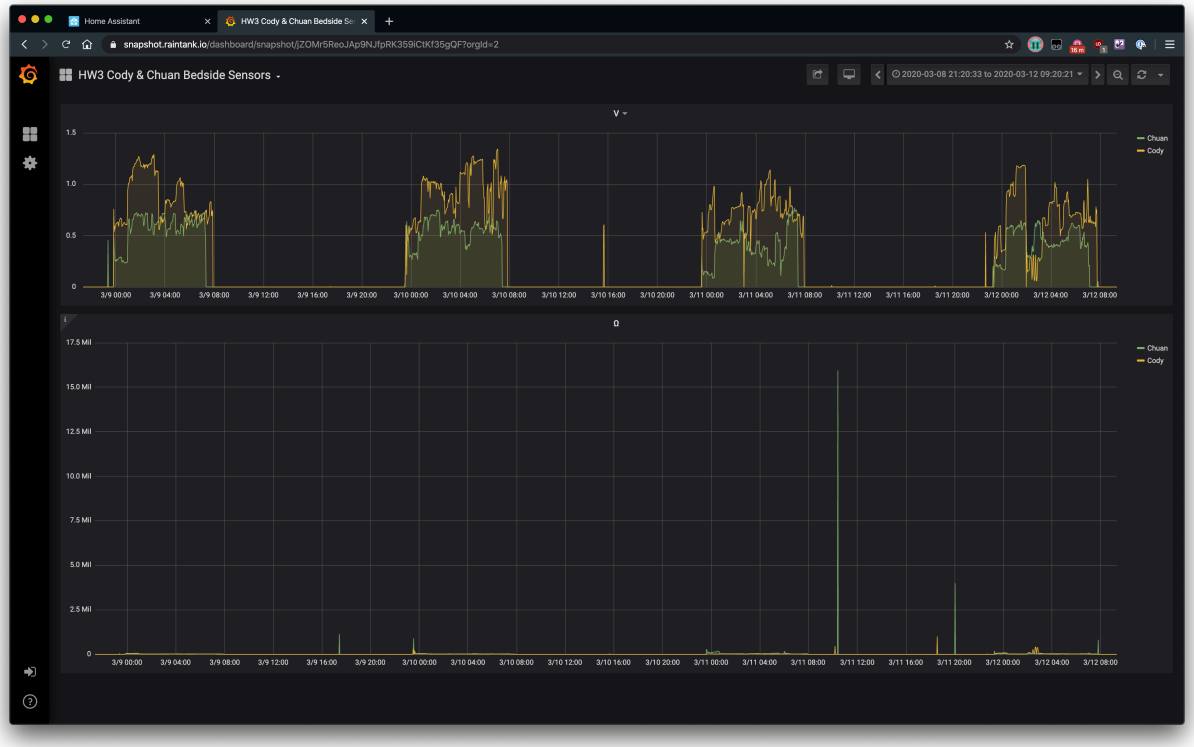


Figure 4: Interactive Grafana Dashboard of the data: http://bit.ly/2QgtOto

Individual days with more data can be found here starting with 3/9/2020: http://bit.ly/2U5oU3o, http://bit.ly/3cZ8an4, http://bit.ly/2U7qjX5, http://bit.ly/2U7JobO

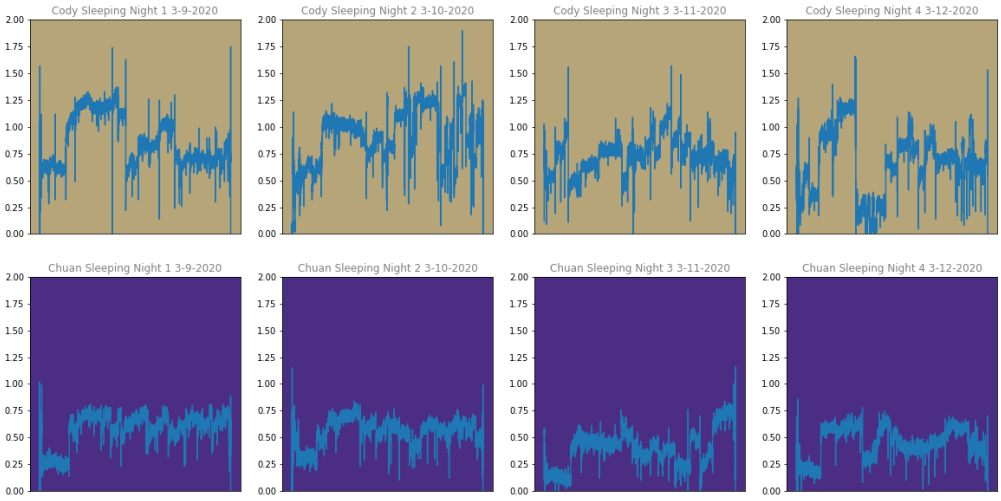


Figure 5: Individual's Sensor Data for Each Night

After exporting my FSR data from influxdb in .csv format, I needed to verify that the data itself was indeed correct! By graphing it I could spot any errors that occurred when I'd exported this data. It also helped to be able to compare the data in this way as it posed a challenge in learning how to graph so many data points (> 192,000 in this small space). Where at first, I was also attempting to graph timestamps, this didn't work due to the closeness of the interval. I removed ticks on this axis and constrained the voltage readings between 0 V and 2 V for interpretability.

On these graphs I noticed that there's a lot more change on my side of the bed with the FSR, which is what I decided to explore for my feature:

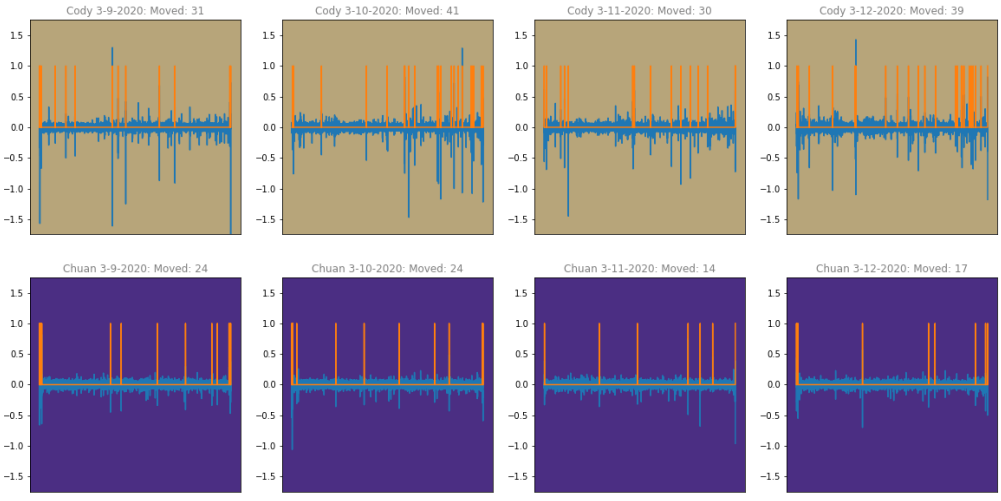
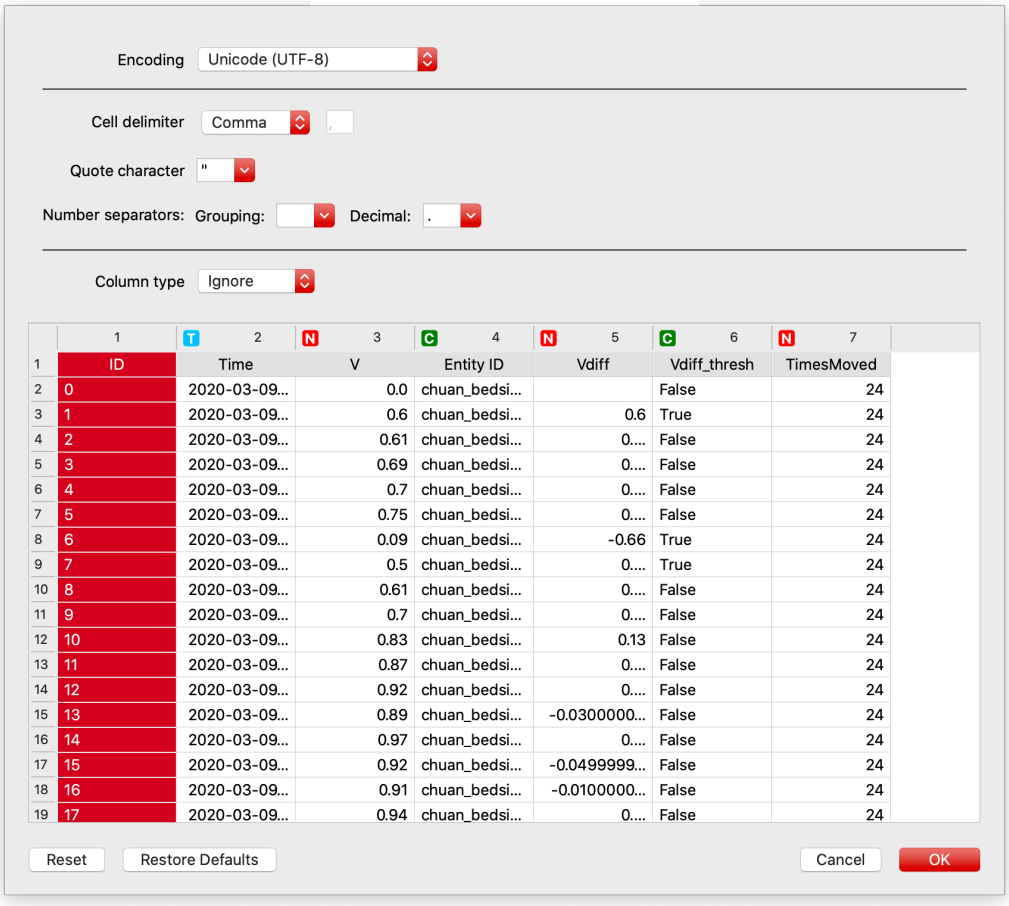


Figure 6: Sensor Data Derivatives (Blue) and Threshold Values for Movement (Red)

I took the difference in each value to see the change from one value to the next, and it turns out this is an indicator of being able to recognize who's occupying the bed while sleeping. We can see from the titles of the graphs the count of movements which were obtained overnight. The blue lines represent graphs of the derivatives while the red lines are based on a threshold. It accounts for positive and negative changes above or below the threshold, respectively. The results are graphed in red from 0 to 1 where 1 is a change which exceeded the threshold. This threshold was ascertained by taking a ratio of my partner's weight to my own weight. I played around with what I thought an appropriate value to determine a change in movement is registered. While a statistical approach can potentially help understand the differences between my partner and I sleeping, I decided the ratio would be much quicker and still provide valid results. While we don't need ML in this example to determine who is sleeping on which side of the bed (we can see from the derivative I move more at night), we could potentially use ML to generalize this behavior and learn about patterns we don't see in this data. Let's find out!

If you'd like to see the code that generated these matplotlib graphs, you can do so here:
http://bit.ly/3aZnNJw



	1	2	3	4	5	6	7
	ID	Time	V	Entity ID	Vdiff	Vdiff_thresh	TimesMoved
2	0	2020-03-09...	0.0	chuan_bedsi...		False	24
3	1	2020-03-09...	0.6	chuan_bedsi...	0.6	True	24
4	2	2020-03-09...	0.61	chuan_bedsi...	0....	False	24
5	3	2020-03-09...	0.69	chuan_bedsi...	0....	False	24
6	4	2020-03-09...	0.7	chuan_bedsi...	0....	False	24
7	5	2020-03-09...	0.75	chuan_bedsi...	0....	False	24
8	6	2020-03-09...	0.09	chuan_bedsi...	-0.66	True	24
9	7	2020-03-09...	0.5	chuan_bedsi...	0....	True	24
10	8	2020-03-09...	0.61	chuan_bedsi...	0....	False	24
11	9	2020-03-09...	0.7	chuan_bedsi...	0....	False	24
12	10	2020-03-09...	0.83	chuan_bedsi...	0.13	False	24
13	11	2020-03-09...	0.87	chuan_bedsi...	0....	False	24
14	12	2020-03-09...	0.92	chuan_bedsi...	0....	False	24
15	13	2020-03-09...	0.89	chuan_bedsi...	-0.0300000...	False	24
16	14	2020-03-09...	0.97	chuan_bedsi...	0....	False	24
17	15	2020-03-09...	0.92	chuan_bedsi...	-0.0499999...	False	24
18	16	2020-03-09...	0.91	chuan_bedsi...	-0.0100000...	False	24
19	17	2020-03-09...	0.94	chuan_bedsi...	0....	False	24

Figure 7: Resultant csv with Vdiff_thresh and TimesMoved features

Moving to Orange, this is the resultant .csv that contains my extracted features. I included the TimesMoved and the Vdiff_thresh features. While I could have used both of these features in ML, I decided to only use the TimesMoved, as it's based off the Vdiff_thresh feature anyways.

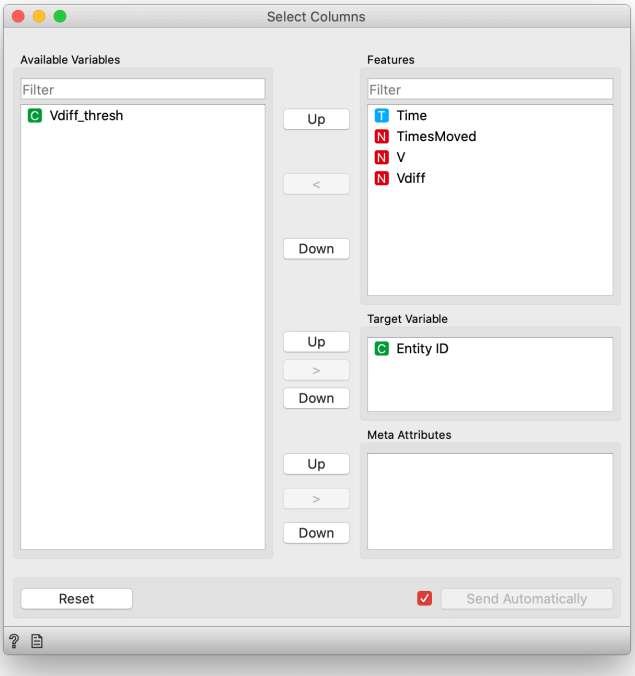


Figure 8: Features chosen for ML

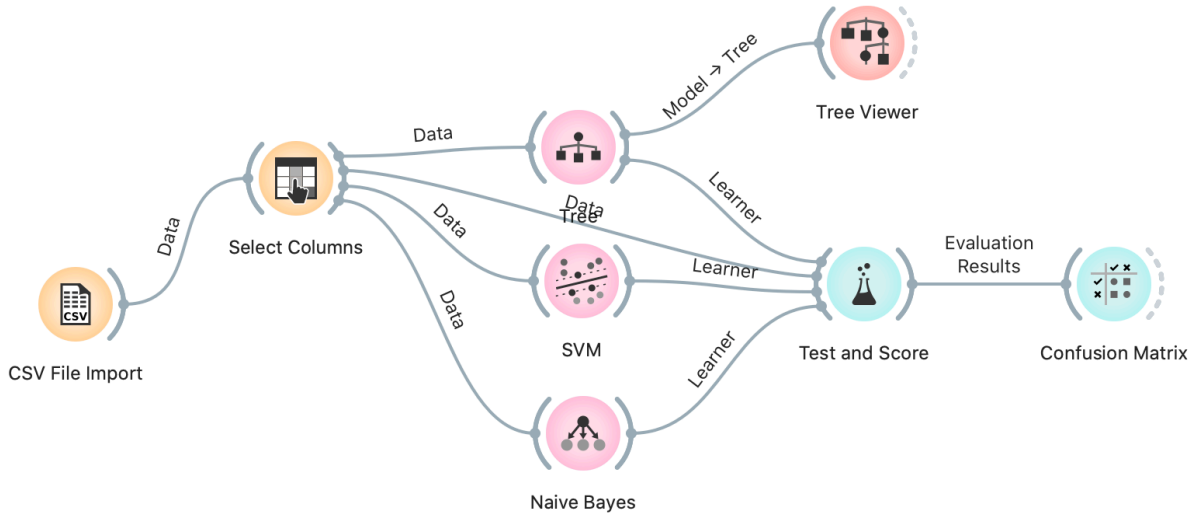


Figure 9: Orange Data Mining Visual Configuration

I used a couple different ML algorithms just to see if anything would perform differently. We have the Decision Tree, Support Vector Machine (SVM), and Naïve Bayes methods.

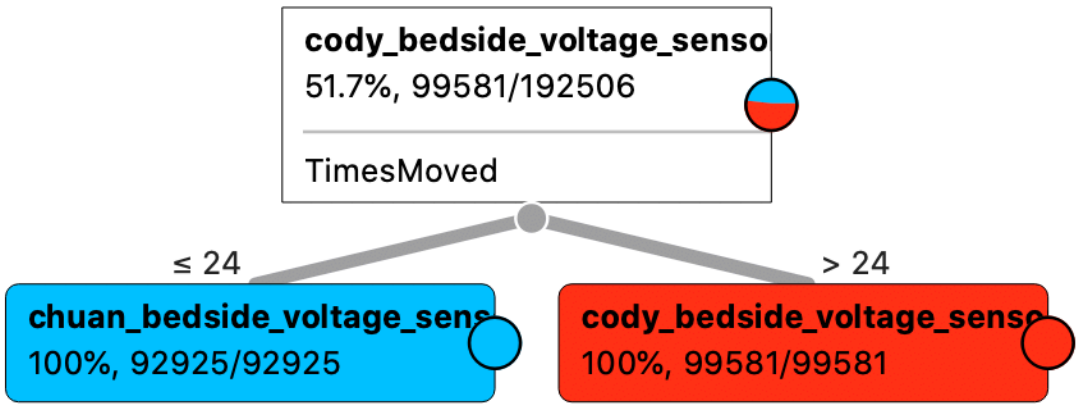


Figure 10: Decision Tree with Extracted Feature

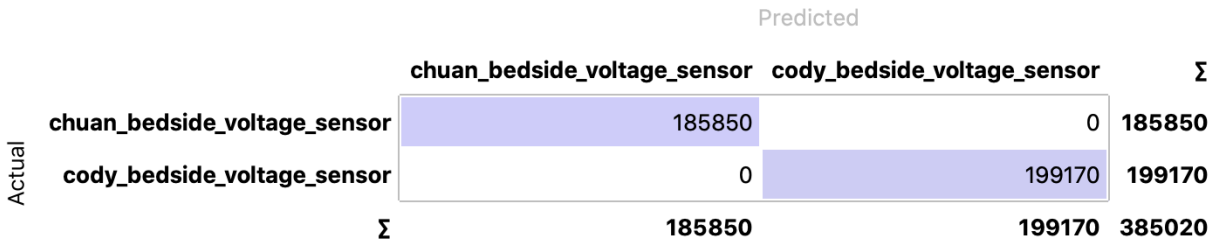


Figure 11: Confusion Matrix with Extracted Feature

Using the extracted feature we ended up with perfectly predicted output using supervised learning! This is unexpected on my part but aligns with the data collected. Following this insight, if there is a statistical method using this sensor we could fingerprint other users and potentially use this to recognize different users based on their overnight movements; since this is only distinguishing between my partner and I though it is currently able to distinguish between us very well. I assume this is because we both sleep differently, and I do move more at night than she does. I am happy that the ML model chose to use this feature for its one and only decision! The only potential drawback I see with this method is that it only counts the number of times moved *after* a night of sleep; a real-time solution that could count the number of movements within a segment of a night and then predict is a fascinating idea.

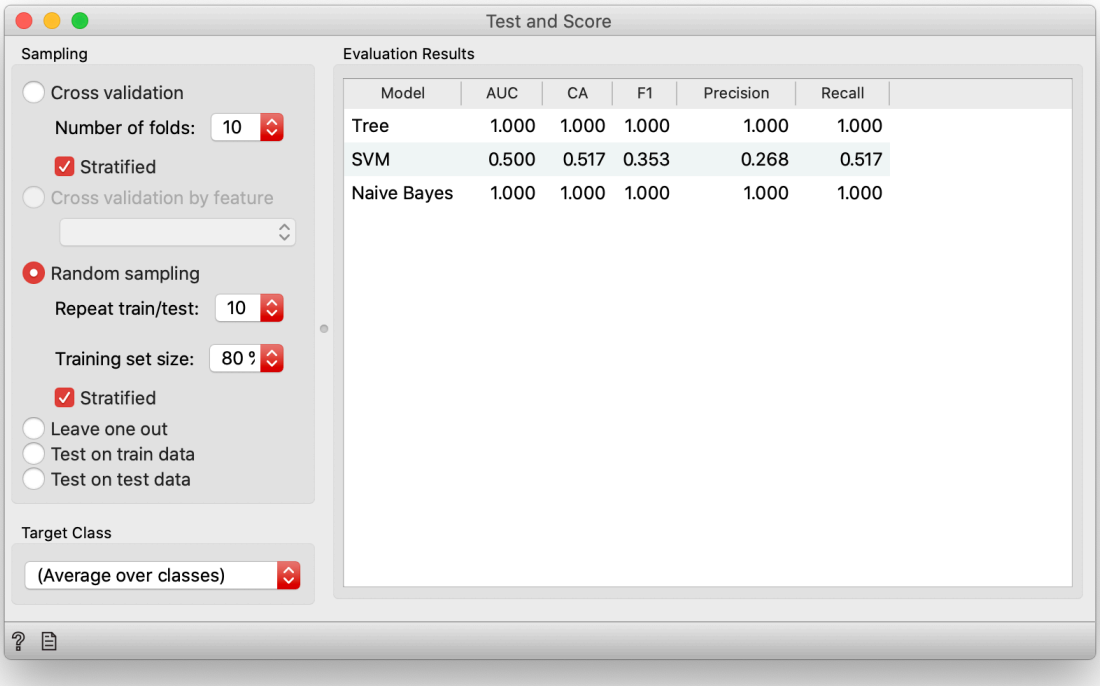


Figure 12: ML Algorithms with Extracted Feature

Not all algorithms were perfect, and SVM did poorly with only a 51% chance of correctly classifying the person sleeping.

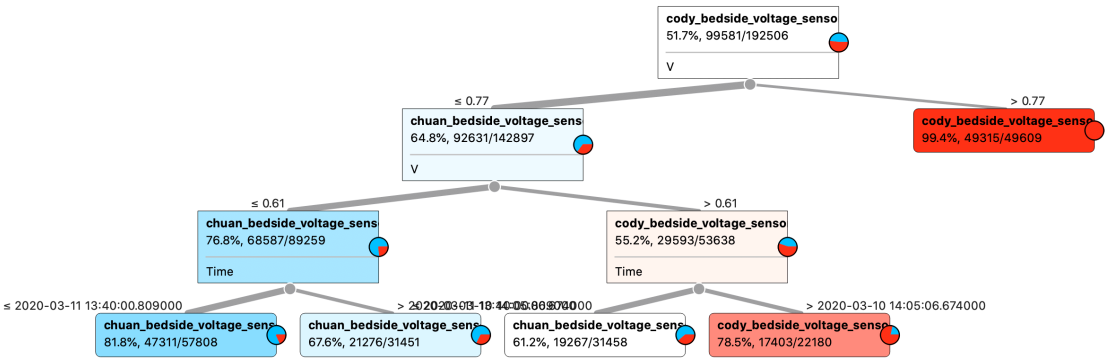


Figure 13: Confusion Matrix without Extracted Feature

Here we see what we've gotten from four days of data without any extracted feature! We have a mix of voltage and time which is used for determining whose data it is. I'm fascinated that time actually could be an indicator and that it was considered by the decision tree. I am glad that

voltage was the first decision used as that is the only actual source of information we are getting from the sensor (otherwise we might be in trouble!)

		Predicted		
		chuan_bedside_voltage_sensor	cody_bedside_voltage_sensor	Σ
Actual	chuan_bedside_voltage_sensor	174368	11482	185850
	cody_bedside_voltage_sensor	64186	134984	199170
	Σ	238554	146466	385020

Figure 14: Confusion Matrix without Extracted Feature

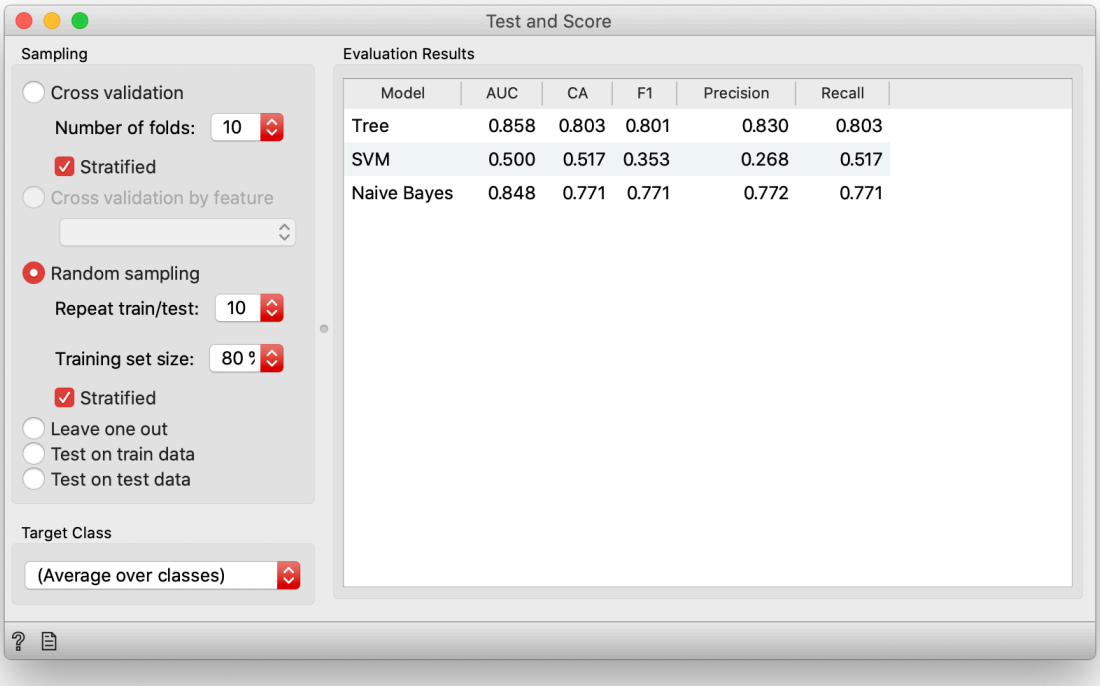


Figure 15: ML Algorithms without Extracted Feature

It is fascinating that the decision three still did really well and got about 80% accuracy without our extracted feature. The Naïve Bayes also did well and SVM got almost the same result as it did with the extracted feature, doing poorly. I believe with predictions this well on the data it was given it may well be possible to create a real-time detector by simply running this model. In terms of generalizability I think more data is required, as it may well be possible that over time our movements change. There are other variables that probably aren't accounted for given this limited window in which this data was measured. There is also another aspect which is comparing these results to other people. This I am not sure of as it requires more sensors and beds to gain the required data. That being said I'm excited seeing ML being able to produce these results and for the things I've learned in this class!