CMPT 353 Project Report:

Gait Pattern Detection

Jeremy Lee 301308486

Behrad Bakhshandeh 301367811

Jiali Cai 301372102

Topic Overview and Top-Down Analysis

Our topic is Sensors, Noise, and Walking, and the problem we are solving is “From a set of walking data, is it possible to distinguish which team member is doing the walking?” It was decided early on to use the linear accelerometer as a measuring tool. As it stands from the raw data, the linear Accelerometer only captures a few things in movement:

- The Timestamp(in seconds)
- Acceleration at that timestamp at the x,y and z axis
- Total Acceleration calculated from all 3 axis:

$$aT = \sqrt{ax^2 + ay^2 + az^2}$$

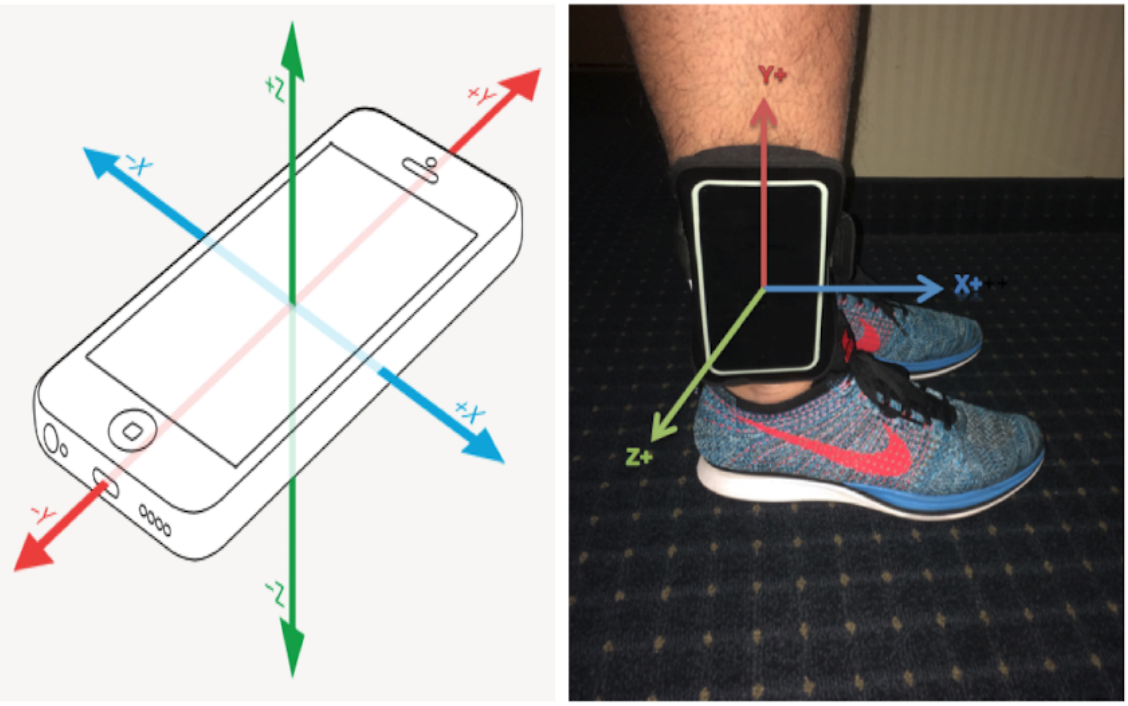
Based on only this information, we thought it would be possible to extrapolate additional information from these measurements(when do we count a step, how many steps were taken, how long did each step last) in order to see if there was a method to distinguish the different types of gait/walking pattern among our team members. This question presents a few issues and other questions:

- How accurate would the data be?
- How do we calculate when a step interval has been completed just based on acceleration?
- How can we verify the results seen here?
- Which Machine Learning Model do we use?
- How accurate would the Machine Learning model be?

It is important to note the main underlying test in our preliminary exploration of this topic: the efficacy of model predictions. The data being collected looked very similar, and was thought to might be extremely difficult for the model to run a proper estimate. This question ended up affecting some of the design choices surrounding the data collection, processing, and evaluation done.

Data Collection

We used the *Physics Toolbox Sensor Suite* that was available for both iPhone and Android devices. We chose to use the Linear Accelerometer, as we initially believed it to contain a more holistic overview of measuring walking patterns. It became a concern of where to place the measuring device, as we originally tested in various places that produced different results. We eventually settled on placing the device outside of our right foot, facing upwards. This would give us the best results, as the location provided maximum movement while walking(the torso/pocket is located around the hip/socket, potentially muting measurements). The image below was provided from a paper covering gait patterns, found here.

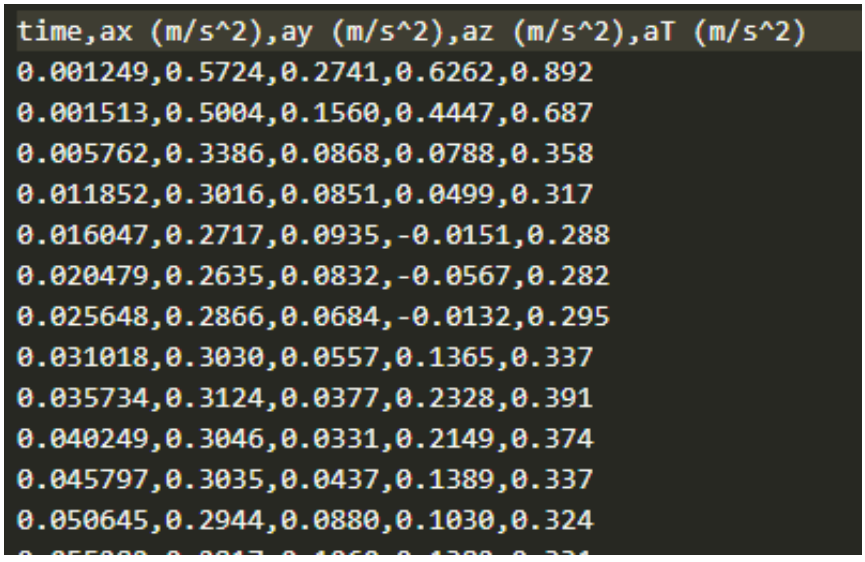


(a) (b)

Figure 3.4 (a) iPhone Orientation (b) gravity vector of the test

The plan was to walk somewhere in which outlying factors such as angle of elevation(walking up a hill), stopping/starting(traffic), the hardness of the surface walking on(grain/sand vs concrete and metal), and the effects of a machine(treadmill) could be mitigated. We decided to use a standard all weather running track(400m). The tracks utilized were from the SFU running track outside the gym, as well as the one outside Point Grey Secondary School. These tracks were beneficial for several reasons; they provided a semi-standard method to be able to loop around, providing data around the same range. The materials used to build the track were also rubberized, allowing for consistent walking across all the test data. It allowed for the above external factors to be mitigated to an extent, and allowed a constant for the data collection to be based on.

From each team member, four laps were measured. Steps were also counted to help verify the step count/measurement process done in the data processing. Three of the laps collected were used to train and build up the model, and the last lap was combined into one dataset for testing the prediction of that model. We made an effort to stop moving at the beginning and end of each measurement so that we could have a clear beginning and ending included in each set. Below is a screenshot of the sample set of raw data that was collected:



```
time,ax (m/s^2),ay (m/s^2),az (m/s^2),aT (m/s^2)
0.001249,0.5724,0.2741,0.6262,0.892
0.001513,0.5004,0.1560,0.4447,0.687
0.005762,0.3386,0.0868,0.0788,0.358
0.011852,0.3016,0.0851,0.0499,0.317
0.016047,0.2717,0.0935,-0.0151,0.288
0.020479,0.2635,0.0832,-0.0567,0.282
0.025648,0.2866,0.0684,-0.0132,0.295
0.031018,0.3030,0.0557,0.1365,0.337
0.035734,0.3124,0.0377,0.2328,0.391
0.040249,0.3046,0.0331,0.2149,0.374
0.045797,0.3035,0.0437,0.1389,0.337
0.050645,0.2944,0.0880,0.1030,0.324
```

Sample Data from Jeremy_555.csv

Butterworth Filtering

The Butterworth Filter was used to denoise the data that we collected. We cut the first and last few rows of data since there is unwanted data where we start/stop walking, in the last step we normalized the data to an interval that revolved around the x axis.

We made experiments of different frequencies for filtering and we found that the frequency with 100 samples/cycle is best to filter our data. The picture below is one of the filtered data with frequency 0.001 (100 samples/cycle), it gives us a smoother plot and does not lose many points.

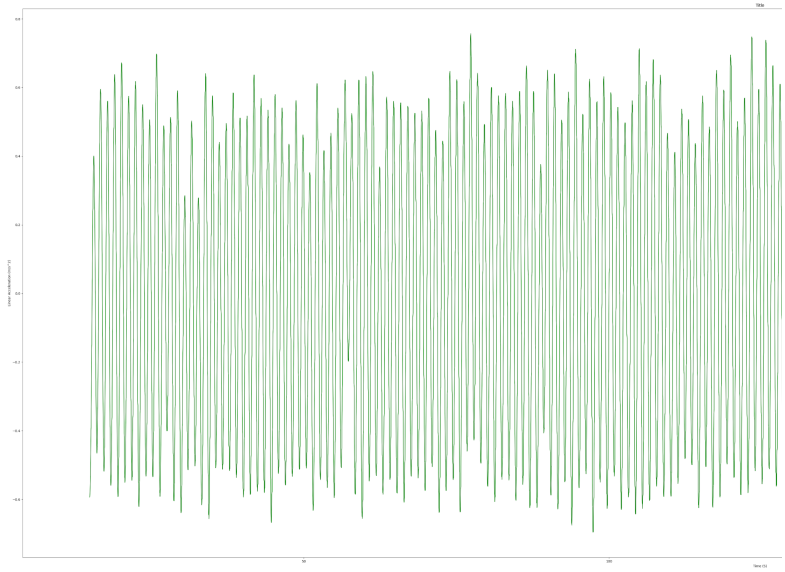


Figure 1: with threshold 0.01 (ideal)

We also tried to filter data with frequency 0.1, 0.2, 0.05, 0.02, and 0.005, but they all look subpar, with some of them having extreme sensitivity to noise after denoise, and some of them losing a great amount of detail after filtering.

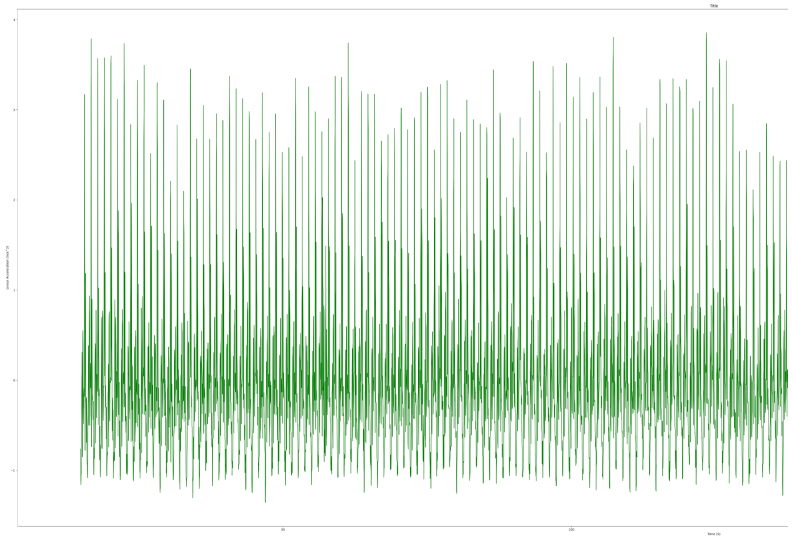


Figure 2: with threshold 0.1 (still noisy)

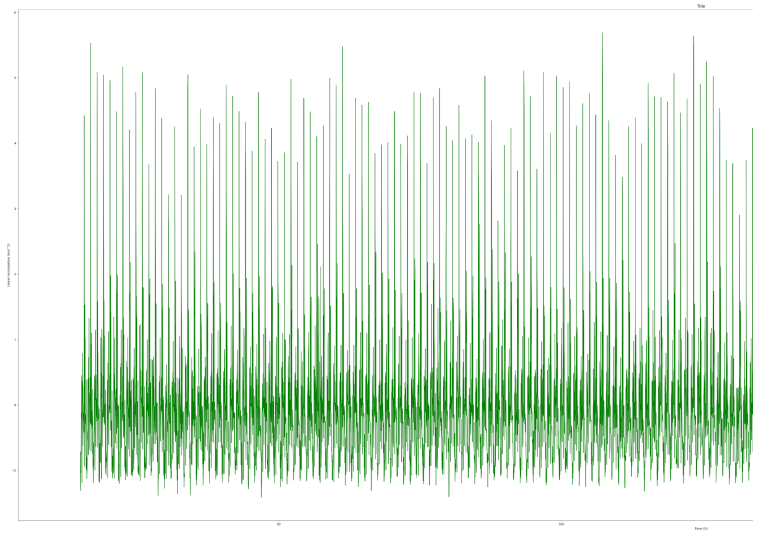


Figure 3: with threshold 0.2 (still noisy)

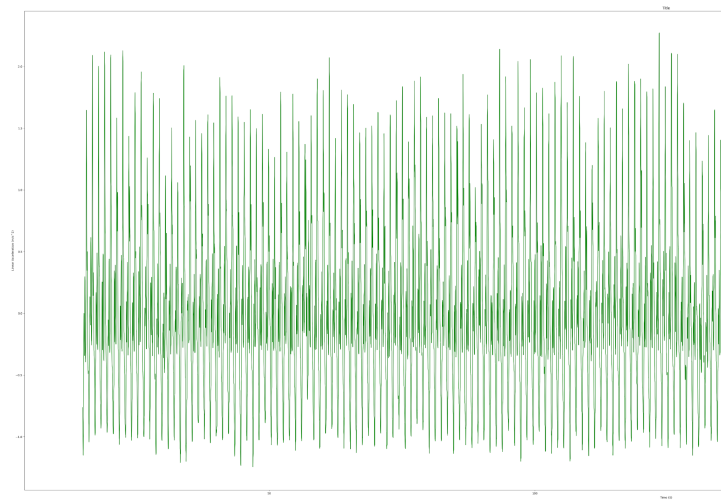


Figure 4: with threshold 0.05 (still noisy)

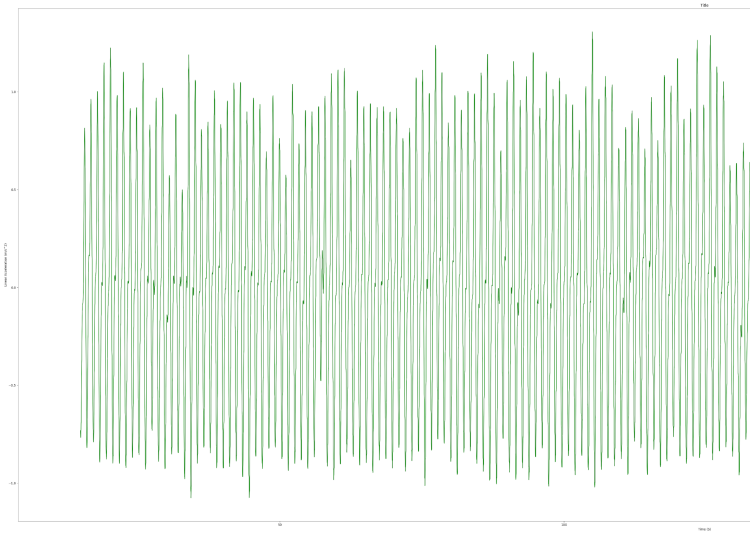


Figure 5: with threshold 0.02 (still noisy)

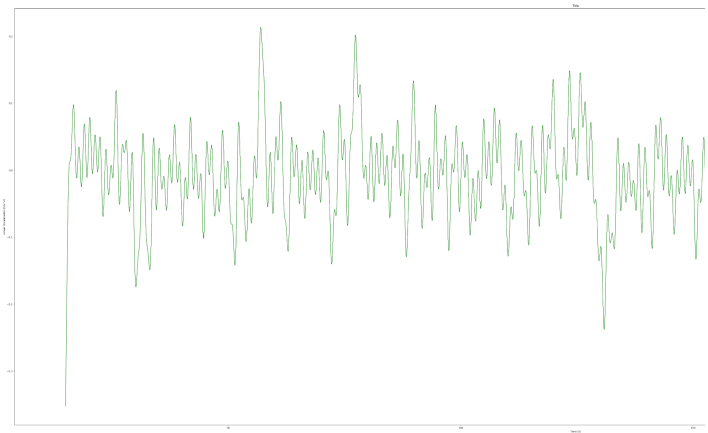


Figure 6: with threshold 0.005 (over filtered)

Normalization

The Normalization was done in preparation of the next step: Step Detection. The formula for the Normalization is provided below:

$$aT_{normalized} = \frac{x-min}{max-min} - \frac{mean(x)-min}{max-min}$$

We begin with a standard minmax normalization to bring the results into a range of [0,1]. Focussing on aT as our primary source of data, we focussed on normalizing aT to a manageable range to extrapolate the steps. We realized that in our walking measurements, aT was moving as a wavelength. This means that in order to calculate the steps, we could wrap our wavelength data around a central point where the majority of the points were skewed towards - and measure every time it crossed that point. That would indicate that a full step has been taken. This is where the last part comes into effect - we take a scaled mean of x, and subtract it, shifting all the data to revolve around 0. The resulting normalized data aT oscillates between 0, making it easier for detecting steps.

It's important to note that while the minmax normalization is a typical method, subtracting down by the mean is not a conventional way of calculating a normalized

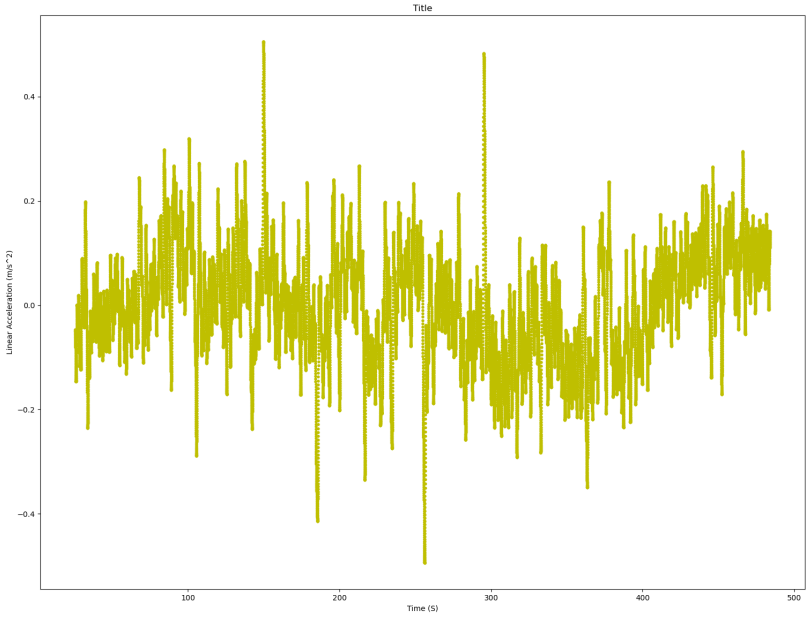
pattern - it was simply a way of manipulating the data to best detect steps in a simplified procedure.

Step Detection:

To be able to have efficient data for our model, one of the attributes we need to calculate is the total number of steps that each subject takes in order to complete the 400m track. Subjects were required to configure their phones in a similar way so that the data gets recorded evenly. Therefore each subject had to strap their phone on their ankle so that the linear accelerometer feature in the app captures the movement of the subject's foot efficiently.

In order to count the total number of steps we looked into the final product of filtered and normalized data. We plotted a graph of total acceleration against time in order to get a better understanding of the movement.

In the following figure we can see the normalized acceleration graph. The range of the data is between -0.5 to 0.5.



In order for our algorithm to successfully identify each step, we took a look at the normalized data and saw how many times it actually crossed the X-axis, each cross would indicate a step and we would count it as one, By performing this algorithm we actually received a 89% percent accuracy compared to the actual steps taken (counted manually), therefore we are confident in our step detection algorithm for each individual data file. The difference from the manual count can potentially be explained due to the trimming done beforehand. The stepCounts variable was appended to our final dataframe in order to be passed on to our model.

The Following Screenshot demonstrates all the attributes that we are passing to our model for training:

```
ormalize
tepAnalysis
     filename        time  ax (m/s^2)  ay (m/s^2)  az (m/s^2)  aT (m/s^2)  aT Filtered  normalized_aT  stepCount
225    Jeremy   15.917239     -0.4196     -0.4103      1.0802       1.229     1.257314      -0.578657          0
226    Jeremy   15.920726     -0.0763     -0.3310      0.9430       1.002     1.268581      -0.577850          0
227    Jeremy   15.925397      0.1448     -0.2301      0.8188       0.863     1.279823      -0.577046          0
228    Jeremy   15.934326      0.2369     -0.1482      0.7531       0.803     1.291038      -0.576243          0
229    Jeremy   15.935604      0.1422     -0.1357      0.6979       0.725     1.302226      -0.575442          0
..         ...         ...         ...         ...         ...         ...          ...            ...        ...
1288   Jeremy  302.554113      2.0330      3.1830     -2.8282       4.718     6.739342      -0.186287        538
1289   Jeremy  302.554611      2.6453      3.5036     -2.9939       5.314     6.794844      -0.182314        538
1290   Jeremy  302.554879      3.1894      3.8097     -3.0903       5.851     6.851685      -0.178246        538
1291   Jeremy  302.555105      3.5366      4.0995     -3.1414       6.260     6.909798      -0.174087        538
1292   Jeremy  302.558683      3.6537      4.3730     -3.3050       6.587     6.969111      -0.169841        538

58068 rows x 9 columns]
ilterAnalysis
ormalize
tepAnalysis
     filename        time  ax (m/s^2)  ay (m/s^2)  az (m/s^2)  aT (m/s^2)  aT Filtered  normalized_aT  stepCount
105    Jeremy   15.324402     -0.0062     -0.0472      0.2906       0.294     0.183276      -0.711815          0
106    Jeremy   15.328954      0.0249     -0.0278      0.2619       0.265     0.182454      -0.711878          0
107    Jeremy   15.333854      0.0701     -0.0655      0.2052       0.227     0.181642      -0.711939          0
108    Jeremy   15.339152      0.1033     -0.0174      0.1490       0.182     0.180841      -0.712000          0
109    Jeremy   15.343936      0.1387      0.0326      0.1005       0.174     0.180052      -0.712060          0
..         ...         ...         ...         ...         ...         ...          ...            ...        ...
9004   Jeremy  291.235921      7.1790      6.8824     -6.8494      12.076     8.835700      -0.054819        552
9005   Jeremy  291.240396      7.4718      7.0395     -7.4718      12.697     8.888359      -0.050820        552
9006   Jeremy  291.245045      7.7282      7.3824     -7.5690      13.096     8.941491      -0.046786        552
9007   Jeremy  291.260982      7.8865      7.8390     -7.2112      13.253     8.995026      -0.042721        552
9008   Jeremy  291.262749      7.8089      8.1841     -7.0735      13.341     9.048897      -0.038630        552

55904 rows x 9 columns]
ilterAnalysis
ormalize
tepAnalysis
     filename        time  ax (m/s^2)  ay (m/s^2)  az (m/s^2)  aT (m/s^2)  aT Filtered  normalized_aT  stepCount
080    Jeremy   15.199636      0.6976      0.8776      0.2265       1.144     1.343629      -0.589215          0
081    Jeremy   15.204197      1.3247      0.8684      0.3453       1.621     1.334901      -0.589837          0
082    Jeremy   15.209393      1.6167      0.1730      0.6506       1.751     1.326079      -0.590466          0
083    Jeremy   15.214135      1.1226      0.2891      0.9550       1.502     1.317171      -0.591101          0
084    Jeremy   15.219272      0.9090      0.6944      1.3238       1.750     1.308181      -0.591742          0
..         ...         ...         ...         ...         ...         ...          ...            ...        ...
8528   Jeremy  288.919837    -13.3007      1.7118     -1.8621      13.539    10.932151       0.094340        503
8529   Jeremy  288.920071    -13.7781      0.4389     -4.5977      14.532    10.894719       0.091671        503
8530   Jeremy  288.921826    -14.4637     -0.4188     -6.3945      15.820    10.856164       0.088923        503
8531   Jeremy  288.927318    -15.6085     -0.9105     -5.9220      16.719    10.816508       0.086096        503
8532   Jeremy  288.932657    -17.6468     -1.1416     -4.1083      18.155    10.775772       0.083192        503

55453 rows x 9 columns]
ilterAnalysis
ormalize
tepAnalysis
     filename        time  ax (m/s^2)  ay (m/s^2)  az (m/s^2)  aT (m/s^2)  aT Filtered  normalized_aT  stepCount
549   Jerrick   25.466678        8.07       -0.89       -0.88        8.16    10.377915      -0.046451          0
550   Jerrick   25.494294        7.89       -0.35       -0.44        7.91    10.374897      -0.047043          0
551   Jerrick   25.496362        7.59       -0.23       -0.28        7.59    10.371484      -0.047713          0
552   Jerrick   25.498045        7.11       -0.31       -0.15        7.11    10.367665      -0.048463          0
553   Jerrick   25.510277        6.61        0.25        0.00        6.61    10.363431      -0.049293          0
..         ...         ...         ...         ...         ...         ...          ...            ...        ...
8431  Jerrick  484.080203       -1.75       -0.35       -0.75        1.93    11.205761       0.115998        295
8432  Jerrick  484.081881       -1.09        0.28       -1.39        1.78    11.198375       0.114548        295
8433  Jerrick  484.090051        0.31        0.73        1.09        1.24    11.191254       0.113151        295
```

A rudimentary view of the data mid-process

Time Intervals and Step Notation

One of the first problems thought of while drafting up the questions was the matter of all the data seeming too similar to one another. As a direct result of this, it was decided to isolate the steps from the other data points. We decided to isolate the rows that the step was detected, and only use those data points for calculation. From there, the row *stepTime* was created to provide the amount of time each step took. This provided a consistent factor of time that could be used in reference to train the model (time by itself would not work as a very good training variable - covered in the next section). All the datasets read from the csv were then appended after all the trimming/processing was done on each file. If appending all the files into one data frame was done first, there would be some issues with the trimming and processing that would ignore the edges of each recorded measurement. The following is a subset of the entire dataFrame after processing.

```
Reading and Processing Data:
      filename        time  ax (m/s^2)  ...  normalized_aT  stepCount  stepTime
2999      Brad   15.220204    -11.8880  ...      -0.001091          1  0.000000
3139      Brad   15.879804      5.5821  ...       0.003398          2  0.659600
3258      Brad   16.440421     -4.3319  ...      -0.000698          3  0.560617
3383      Brad   17.029245     10.7640  ...       0.001465          4  0.588824
3500      Brad   17.580682     -8.7918  ...      -0.001069          5  0.551437
...        ...         ...         ...  ...            ...        ...       ...
44183  Jerrick  441.641555     10.6300  ...       0.002696        333  0.461810
44913  Jerrick  448.936681     -0.2600  ...      -0.001459        334  7.295126
44937  Jerrick  449.177158      0.8600  ...       0.001496        335  0.240477
45123  Jerrick  451.035835    -23.9800  ...      -0.003703        336  1.858677
45195  Jerrick  451.761217     12.1900  ...       0.000607        337  0.725382
```

All Formatted Data

The column *filename* was simply extrapolated from each of the filenames titled with our own names in a similar naming scheme.

Column Removal and Model Preparation

Some of the Columns that were in the current dataFrame would interfere with the model. The biggest issue was with the *time* column, as that column has similar values across all csv files. The data that should go into the models should be varying, not similar. As a result of this, we removed the *time* column, as well as the *aT (m/s^2)* and *filename* column (the *filename* column was transferred to the y_train structure as classifier data). The following is the final data for X:

Modelling Data:
X_train structure:

	ax (m/s^2)	ay (m/s^2)	az (m/s^2)	aT (m/s^2)	normalized_aT	stepTime
22761	12.9361	2.3201	8.7214	15.773	0.000921	0.520679
21561	-0.4300	0.0300	-1.9200	1.960	-0.002835	0.886553
19413	-31.1500	5.0700	6.9000	32.300	-0.000301	0.220273
5618	0.8618	-0.2344	3.3945	3.510	-0.000888	0.089553
45728	2.3825	1.4338	-1.3343	3.084	0.002173	0.148482
...	...	...	...	...	...	...
38440	2.2229	-3.3110	0.7637	4.060	-0.000069	0.157489
23780	-1.8455	-1.3277	1.7941	2.896	-0.000141	0.632007
51247	-2.8746	-1.6422	1.6633	3.705	-0.001791	0.628954
51776	7.2434	-0.7491	-1.2861	7.395	-0.007342	0.604599
16645	-4.1249	-0.5549	0.5582	4.199	-0.002079	0.600170

Data for the X_train Structure

Model Training: Picking the Strongest Model

We used eight different models for training the data. The models are:

- Gaussian Naive Bayes
- K-Nearest Neighbours (4 Neighbours)
- K-Nearest Neighbours (7 Neighbours) - More Neighbours
- K-Nearest Neighbours (13 Neighbours) - Most Neighbours
- Random Forest (30 Trees, max depth of 4)
- Random Forest (50 Trees, max depth of 4) - More Trees
- Random Forest (30 Trees, max depth of 8) - More Depth
- Random Forest (50 Trees, max depth of 8) - More trees + Depth

For each of these models, the training data is fit inside, and score calculated from the valid data. From there it chooses the model with the highest score, and uses that model as the predictor for that iteration. For a full log of everything, please visit the *outputlog.txt* file.

```
Bayes Model:
0.5875831485587583

KNN Model:
0.8159645232815964

KNN with more Neighbours:
0.8170731707317073

KNN with most Neighbours:
0.8093126385809313

RF Model:
0.7439024390243902

RF with more trees:
0.7416851441241685

RF with more depth:
0.8636363636363636

RF with more Trees and Depth:
0.8603104212860311

MODEL WITH THE HIGHEST SCORE TO BE USED:RF with more depth
```

An example of outputlog.txt - evaluating the strongest model

Model Prediction and Accuracy Evaluation

The Strongest Model passed from the Model Evaluation is then provided with test data from some sample data to assess the overall accuracy of the model. The data in the *Predict* Folder was read, formatted, and concatenated in a matter similar to the original input training data. The data at the end is similar to the X_train dataframe seen above. The data is then calculated, and the model does its best to predict whose gait pattern is whose. The results are then outputted into *output.csv*, in which the predicted values are

compared to the actual values, and an accuracy score is printed out(can be seen in *outputlog.txt)*

```
Testing predictions
     predicted_values actual_values  is_correct_prediction
0              Jeremy          Brad                  False
1              Jeremy          Brad                  False
2                Brad          Brad                   True
3                Brad          Brad                   True
4                Brad          Brad                   True
...               ...           ...                    ...
1301          Jerrick       Jerrick                   True
1302          Jerrick       Jerrick                   True
1303          Jerrick       Jerrick                   True
1304          Jerrick       Jerrick                   True
1305          Jerrick       Jerrick                   True

[1306 rows x 3 columns]

TOTAL CORRECT: 958 OUT OF 1306 : 0.7335375191424196% CORRECT
```

An example of the verification data

Result Expectations

One of the biggest issues mentioned previously and throughout was the worry of overlapping data. As measures such as acceleration were considered, there was a lot of room for overlap. As such, we did not have very high expectations regarding the model scoring, or the verification of accuracy that followed.

While processing the data, however, our expectations shifted. By carefully planning the execution of data collections, and ensuring that our remaining data frame contained only important values that wouldn't sway the model falsely, we were able to feel fairly confident in the model, albeit still unsure. We estimated that roughly we would get a **55-65% accuracy** on the model (getting a little more than half right).

Result Accuracy

After successfully training the models we had to evaluate it by passing on our test data files that we separated earlier, and tested the score of our models.

Model	Score
Bayes Model	0.598669623059866
KNN Model	0.802660753880266
KNN with more Neighbours	0.809312638580931
KNN with most Neighbours	0.802660753880266
RF Model	0.741685144124168
RF with more trees	0.750554323725055
RF with more depth	0.845898004434589
RF with more Trees and Depth	0.856984478935698

Based on the results we can see that RF with more trees and depth is the highest scoring model in this scenario and therefore it would be logical to use it as the main model for our problem. However, through multiple iterations of running the data, there were slight variations in scoring and even in the strongest model picked. About roughly 30% of the time we ran *dataProcessing.py*, we found that the highest score would actually be *RF with more depth,* and the other 70% of the time the highest score would be *RF with more Trees and Depth*. Consistently, it was clear that depth in RF was one of the primary factors in our model in determining a higher score.

This does make sense, as more complex trees in a forest would allow us to make more nuanced/complex predictions in classification assessment. Having a deeper tree would allow even smaller differences in values to contain more weight in evaluation. In this particular iteration above, notice the difference between *RF Model* and *RF with more trees* vs *RF Model* and *RF with more depth*. Having more trees does not necessarily improve the model on itself. The Random Forest *n_estimators* is the subset of trees that

typically give a closer holistic picture to the value, but don't improve the actual decision making of a single tree and node traversal.

Additionally, for every iteration, the model predicted the values with a **70-75%** success rate. For a full visualization of data, please see *output.txt*. This was much higher than what we initially expected, potentially due to all the noise filtering and data processing done beforehand to isolate distinct values. A higher-than expected score could also be attributed to the amount of values being run into the model for training - more variations in different variables leads to a stronger, more accurate model.

Limitations

One limitation is the devices that we used to collect data. The sensors in our phones are different and we are relying on a third party app to collect data from the sensor for us. We think a professional device will provide us more accurate data. Having an external sensor like a phone introduces other issues as well. If the device is not securely fastened, it creates its own force/acceleration vectors relative to the leg, swaying results measured.

Additionally, there was no method to trigger the application recording process. We had to trigger it, place the phone in position, wait, and then begin to walk. The opposite happened when stopping the timer. This created less than standard points of garbage data. Without a proper way to detect where to trim, the current trimming procedure does cut off some of the non-garbage data, shortening the training pool for the model.

The runtime of this program is exceptionally slow. Due to the nature of having to read the file in, perform dataframe operations, and append them into an array for each .csv file. This action of actively reading in the files like this is the slowest part, with bottleneck happening in the reading itself.

Conclusion

Originally we thought that a model for this type of case wouldnt accurately predict whose gait/walking patterns were being displayed. We managed to mitigate these worries through a combination of proper data collection practices, trimming, filtering, normalization, step detection, creating the time for each individual step, removing unneeded columns, and picking the model with the highest score, we were able to drastically improve our expectations of the models prediction. By using a RF Model prioritizing depth(and potentially increasing the amount of trees for a more accurate result), we are able to reasonably predict which team member is walking from a Linear Accelerometer data set.

Project Experience Summary

Jeremy: “Created a Data Model Evaluation and accuracy assessment pipeline to reliably choose and test the strongest Model’s predictions.”

Brad: ”Created an algorithm for step detection and time step calculation and appended it to the final dataframe.”

Jerrick: “Created a pipeline to trim, filter, and denoise data so that the data we used for step counting/training is tidy and clean.”

References

- Gitlab clone: https://csil-git1.cs.surrey.sfu.ca/jwl45/cmpt-353.git
- Gait Pattern Dissection: http://summit.sfu.ca/system/files/iritems1/17204/etd10035_MYousefian.pdf