Fall Detection: Using Linear Acceleration Data

CMPT 353: Computational Data Science

Professor Greg Baker

Summer 2020

Joshua Leung - 301392538

Pruthviraj Patel - 301300329

Harnoor Singh - 301355738

TABLE OF CONTENTS

1. Introduction

Some seniors are at a higher risk of falling than others. Most traditional medical alert monitoring systems rely on a button that your loved one must press to call for help. Although these systems are highly effective and save lives every day, there is the possibility that when your loved one falls, they may be unable to press their medical alert button to call for help.

In this project, automatic fall detection with data science is explored. The idea is that when a phone is dropped and hasn't been picked up for an extended period, an emergency call is automatically activated. The scope of the idea is too complex to finish within one month; therefore, it was focused on a small part of it -- using machine learning to identify a fall.

The objective of the project is as follows:

- Gather digital signal data with the iOS app called Sensor Data Recorder
    - Only five groups of data were gathered: Walk Sit, Walk Fall, Down Sit, Run Fall and Free Fall. The details will be discussed in Section Two.
- Clean and transform digital signal data
- Explore the processed data to discover features
- Apply different machine learning models to the data and choose the most suitable one(s)
- Analyze different groups of data with a few statistical tests
- Predict new data with the trained machine learning models
- Present the results

2. Data Acquisition

The data for the problem that was to be solved was available on Kaggle, and GitHub had some data sets(s) as well that could be used. However, all data set(s) that were found had columns that were not required. So, it was decided that new data would be collected. An application was used on our phones called “Sensor Data Recorder” by Nils Ackermann to collect the data. This application is available on the iPhone and was perfect for our data acquisition. The data sets were recorded for various scenarios. Below describes each one in more detail:

2.1 Walk Sit

For this scenario, a subject would start the recording and walk at a reasonable pace, having average arms and feet movement for ~45 seconds and then, it would transition into a sitting action where they would either sit on a couch (or similar) or on the floor, stopping their movement. As well, the recording will be stopped. While they are walking, how and where they walk, i.e. walk in a straight line, on a crosswalk, curved path etc. does not matter. For ease of convenience, the individual just decided to walk and transition into sitting in their own house.

2.2 Walk Fall

In this, a subject was considered who would start the recording and walk at a reasonable pace, having average arms and feet movement for ~45 seconds and transition into a falling action (or act as if they had been tripped). After that, they would swiftly drop their phone on a pillow (or something similar to ensure they do not damage/break their phone) and stop the recording after the drop. Again, where and how they walk, whether it be in a straight line, on a crosswalk, curved path etc. does not matter. For ease of convenience, the individual just decided to walk, transition into falling and drop their phones (on a pillow or some cushion) in their own house.

2.3 Down Sit

For this scenario, a subject would walk at a reasonable pace, having average arms and feet movement for ~30 seconds after starting the recording and then would go down the stairs as one would typically at a reasonable, steady pace. The individual would continue going downstairs for ~5 seconds, come to a halt, sit down on the stairs and stop the recording. Again, this was done at that individual's house for ease of convenience. Note that how they walk initially, i.e. straight line, curved path etc. does not matter.

2.4 Run Fall

In this scenario, after starting the recording, an individual would run again for ~45 seconds at a brisk pace, having average arms and feet movement as they would while normally running. They would then transition into a falling action (or act as if they had been tripped). After that, they would swiftly drop their phone on a pillow (or some cushion) and stop the recording. Where and how they run does not matter. Here, the subject just chose to do this at their place for convenience.

2.5. Free Fall

Lastly, relative to where they plan on dropping their phone, the individual will stay ~4.5m above "ground." They would have put their phone in a bag that has a lot of cushion support after starting the recording. Then, they would drop the bag with the phone from a height of ~4.5m. There would also be a person standing by to where they are dropping the phone. The receiver would open the bag after the drop to stop the recording. Again, this was done at the subject's place for ease of convenience.

Note :

1. The reason to record similar actions was if there was any significant difference between the activities involving fall and other related activities and if we could differentiate between them.

2. All three members of our group collected data. So, It could notice if any of our phone sensors were recording data differently.
3. The main goal of the project was if it was possible to differentiate between fall and not fall with given data. This implies it to be a classification problem.

3. Data Exploration

Data exploration is the initial step in data analysis. Here, data was initially loaded, and six pandas data frames were created, one for each scenario described above. These data frames had lots of rows since data for each situation was recorded many times (it is mentioned in the Data Acquisition part), and each recording was just appended to their respective data frames for each scenario. They each had seven columns, with the notable (useful) ones to us being, AccelerationX, AccelerationY and AccelerationZ.

3.1 Analysing data

After loading data into six data frames, one for each scenario, the accelerations of those data frames were plotted, and it was noticed that there was a lot of noise that our sensor had picked up. So, before that data could be used for analyzing, it had to be ensured that it had as little noise as possible and to solve this problem, a filter had to be decided that could be used to minimize noise.

3.2 Cleaning data

A few different filters were tried on the accelerations for each scenario, namely Kalman filter, Lowess Smoothing, high-pass filter, low-pass filter, Fourier transform filter etc. However, the filter that gave us the best results was the Butterworth low-pass filter. Using this filter on columns AccelerationX, AccelerationY and AccelerationZ for each scenario, five new data frames were created. With all the columns present in

the original data frames, the filter was already applied to all accelerations for each scenario. Below is a graph of one of the situations that compares original data to the data after the Butterworth filter was applied to AccelerationX, AccelerationY and AccelerationZ. The results for before and after cleaning the data are in figure 3.1.

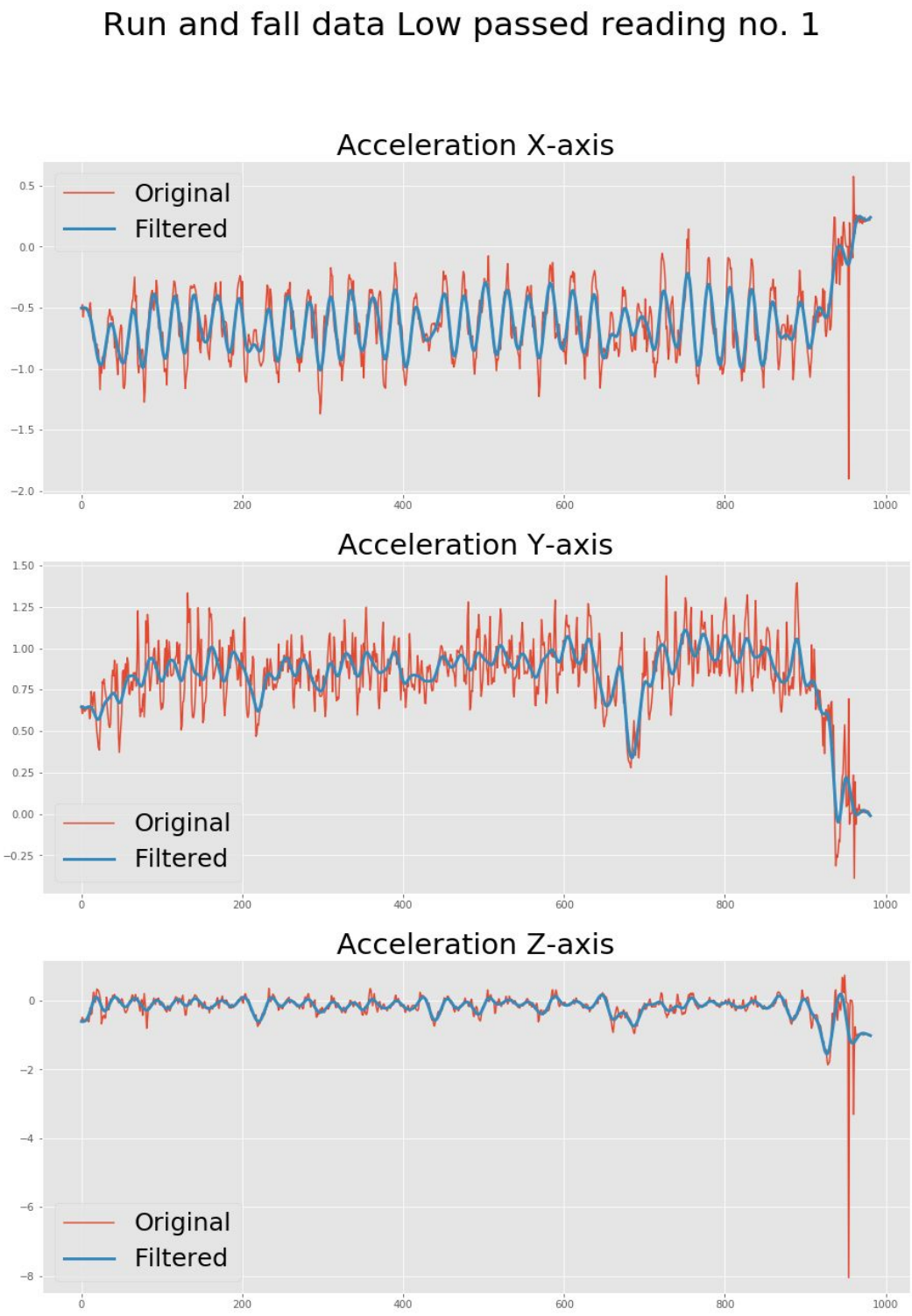


Figure 3.1: Original and Cleaned data

3.3 Total acceleration

The primary focus was to use just the Linear acceleration data to make the predictions. Data had acceleration along X-axis, Y-axis, and Z-axis. To calculate the total acceleration linear acceleration formula was used, and then a new column with total acceleration was added to all the data frames.

$$A_t = \sqrt{A_x + A_y + A_z}$$

Figure 3.2: Total Acceleration formula

4. Statistical Analysis

4.1 Line Plot

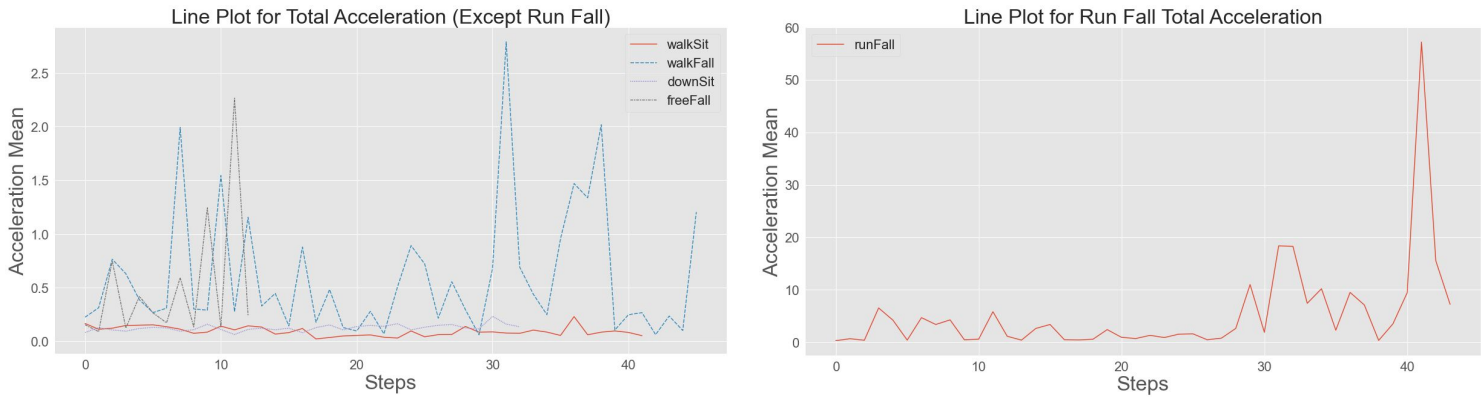


Figure 4.1: Line Plot for Total Acceleration of 5 Groups

The above figure depicts the total acceleration of all groups on a line plot. Run Fall has higher total acceleration compared to the other four groups. In those four groups, Walk Fall dominates the chart while Walk Sit stays at the bottom.

4.2 ANOVA and Post Hoc Analysis with Tukey's HSD test

ANOVA was applied to the data to determine if the means of any of the groups differ. The P-Value of the test is 5.068597522945221e-07, which suggests that there is a difference between the means of the groups.

Next, it was desired to find out how the groups differ in pairs. Through Post Hoc Analysis with Tukey's HSD test, pairwise comparisons between each variable were performed.

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
downSit	freeFall	0.3802	0.9	-3.8185	4.5789	False
downSit	runFall	5.1699	0.001	2.2172	8.1227	True
downSit	walkFall	0.4738	0.9	-2.4513	3.3989	False
downSit	walkSit	-0.0324	0.9	-3.0151	2.9504	False
freeFall	runFall	4.7898	0.0115	0.7421	8.8374	True
freeFall	walkFall	0.0936	0.9	-3.9339	4.1212	False
freeFall	walkSit	-0.4126	0.9	-4.4822	3.657	False
runFall	walkFall	-4.6961	0.001	-7.4	-1.9923	True
runFall	walkSit	-5.2023	0.001	-7.9684	-2.4362	True
walkFall	walkSit	-0.5062	0.9	-3.2427	2.2304	False

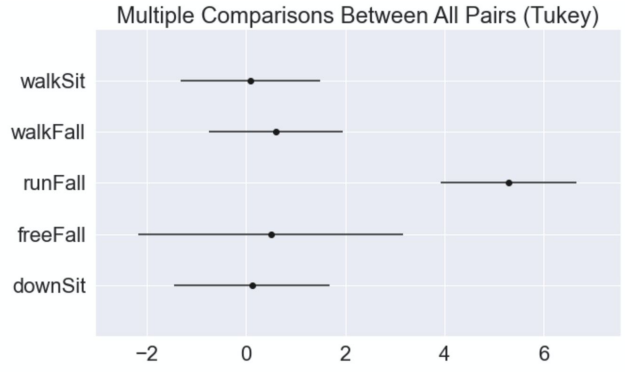


Figure 4.2: pairwise comparison between each variable

The table was observed, and a spiffy plot was produced by the post hoc Tukey test. It was suggested that Run Fall has different means with all other groups; for other pairs, it cannot be identified. This is not surprising because Figure 4.1 indicates the same idea.

5. Feature Engineering

The actions with the lowest total linear acceleration were for the Walk Sit and Down Sit. In general, the peak of the highest total acceleration was between 2.0 m/s^2 and 7.0 m/s^2 for sitting situations. A possible reason is that sitting is a controlled motion with a reduced acceleration.

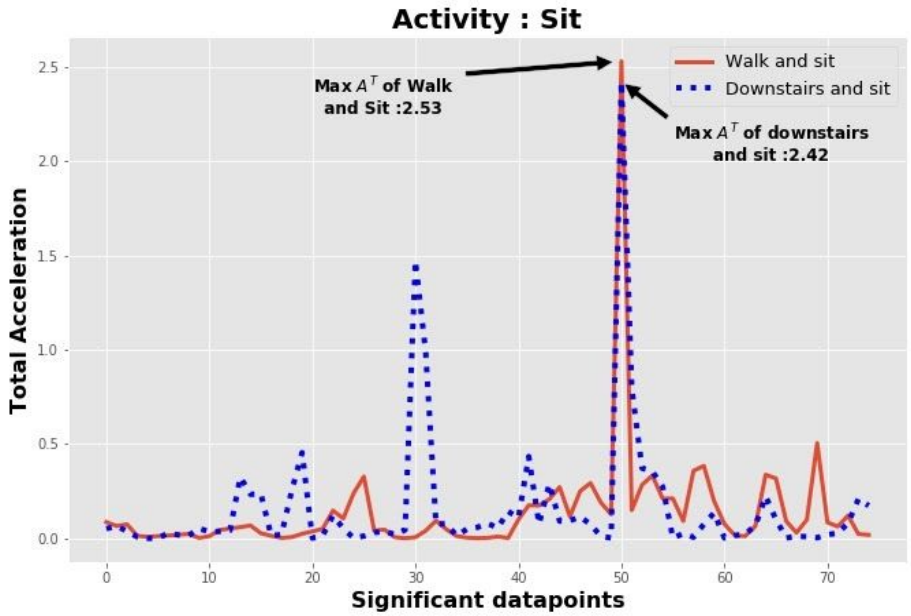


Fig 5.1: Free Fall Significant data points

The following are both the fall situations, consisting of the Walk Fall, and Run Fall actions. In general, the peak of the highest total acceleration was between 15.0 m/s² and 60.0 m/s² for fall situations. It is likely due to the uncontrolled range of motion while falling.

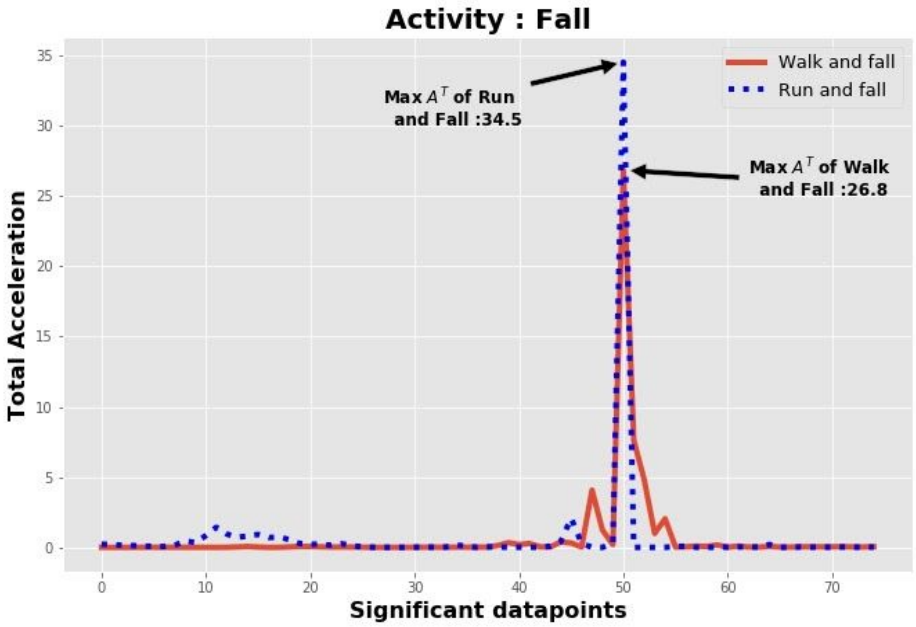


Fig 5.2: Free Fall Significant data points

The Free Fall situation had the highest total acceleration out of all the actions. Upon observation, the peak of the highest total acceleration was between 70.0 m/s² and 120.0 m/s².

The Free Fall action involves increased velocity for a long time with little or no resistance resulting in peaking total acceleration.

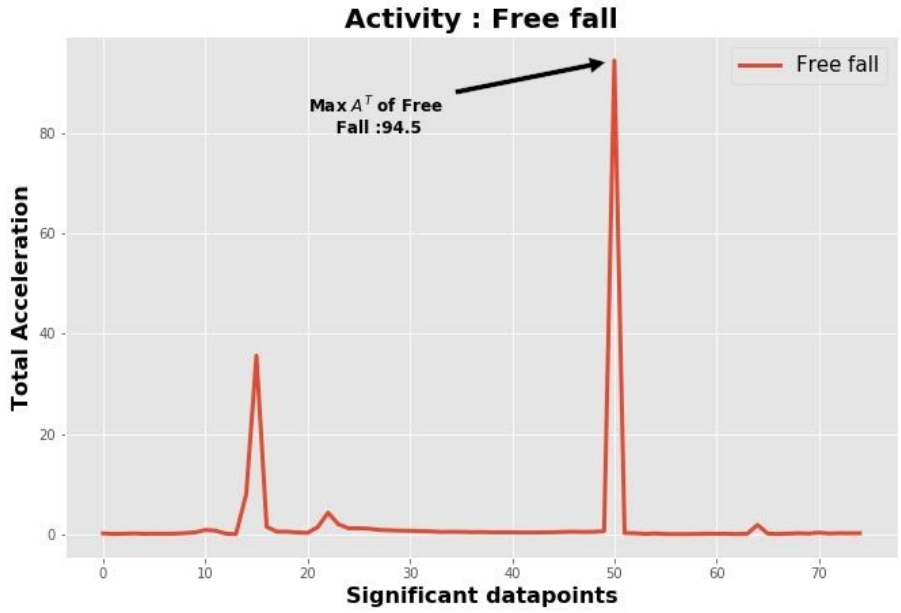


Fig 5.3: Free Fall Significant data points

Most of the data had nominal acceleration before the actual action took place, So data trimming was required. After many trials, a range of -50 and +25 data points before and after the maximum total acceleration respectively were chosen, and every data frame was trimmed according to that window. After trimming the data, much clearer peaks were visible, and the change in acceleration was easily noticeable in the total acceleration graph.

After trimming each dataset, the mean of each acceleration was calculated, and a new data frame for each situation was created. So initially, for training the model, we had the mean X-axis, Y-axis, Z-axis, and total acceleration for each recording of distinct actions. The results were not as expected, and the scores were deficient.

Later, a forward attribute selection technique was used until an accuracy to the plateau was achieved. In which columns with Minimum, Maximum Value and Standard deviation for each of the X-axis, the Y-axis, Z-axis, and total acceleration were added. After adding each row one by one, a model was trained and tested until we received some significant results.

A target column was required for machine learning. Hence a column ‘Fall’ was added to each dataset, which depicted if the features in that row are of a fall or not.

	MeanX	MeanY	MeanZ	MeanT	StdX	StdY	StdZ	StdT	MinX	MinY	MinZ	MinT	MaxX	MaxY	MaxZ	MaxT	Fall
0	0.109075	-0.983321	-0.303333	2.789346	0.136728	0.416518	0.367934	22.491346	-0.075117	-2.024085	-1.073098	0.000140	0.483072	-0.488293	0.618363	194.902482	1
1	-0.610111	0.789183	-0.049740	0.105961	0.052761	0.068138	0.084297	0.183009	-0.761631	0.635520	-0.263401	0.000040	-0.515161	0.904446	0.061768	0.942483	0
2	0.762375	0.186843	-0.255610	0.689832	0.436812	0.304532	0.448658	3.817050	-0.037034	-0.384185	-0.943642	0.000017	1.229279	0.730881	0.723570	32.711325	1
3	0.310770	0.162884	-0.960604	0.054336	0.058925	0.039333	0.039909	0.036482	0.216099	0.083330	-1.040035	0.000365	0.479804	0.238537	-0.887886	0.221693	0
4	-0.551340	0.375405	-0.408236	1.132038	0.403998	0.683267	0.385454	7.524527	-1.287443	-0.420773	-0.899816	0.000006	0.112294	1.650612	0.207007	65.276002	1

Fig 5.4: Final data frame

6. Machine Learning

With the results from feature engineering, we had a total of 16 feature columns that were used to predict the target, i.e. if features in that row depict a fall or not.

6.1 Training models

In total, 11 different models were used, several manual runs were done with different hyperparameters, and with the best parameters, separate functions were made for each. Below is the list of all the different classifier used:

1. Voting Classifier
2. Random Forest Classifier
3. Support Vector Classifier
4. K-nearest neighbours Classifier
5. PCA with Support Vector Classifier

6. Decision Tree Classifier
7. Naive Bayes Classifier
8. Naive Bayes Classifier with prior scores
9. Gradient Boosting Classifier
10. AdaBoost Classifier
11. Multi-layer Perceptron classifier

6.2 Evaluating different models

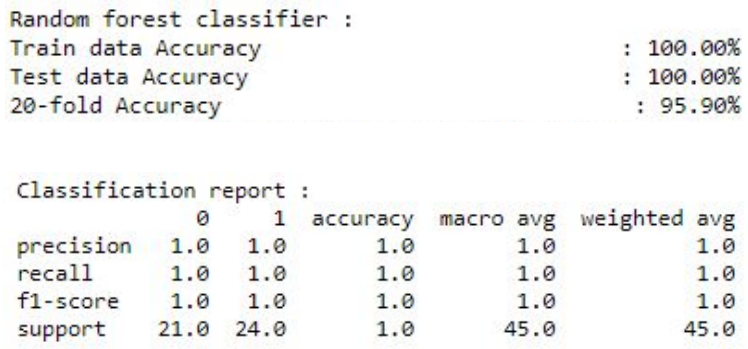
After training the models, a profound evaluation was to be done to determine the results of different models. Initially, the validation technique used was just a score-method on the validation data. But, it didn't sound right because one could be while splitting the data into train and test. So, a cross-validation score was used to train and test all the models on 20 distinct folds of the data, and a mean of the test scores was taken to determine its accuracy.

Then a Receiver Operating Characteristic (ROC) Curve was used for further analysis. ROC is a comparison between the true positive rate(tpr) and false positive rate(fpr). In this case, it means:

- True-positive = model predicts Fall when the truth is Fall
- False-positive = model predicts Fall when the truth is Not a Fall

The models with the best results are :

1. Random Forest Classifier



Random forest classifier :
Train data Accuracy : 100.00%
Test data Accuracy : 100.00%
20-fold Accuracy : 95.90%

Classification report :

	0	1	accuracy	macro avg	weighted avg
precision	1.0	1.0	1.0	1.0	1.0
recall	1.0	1.0	1.0	1.0	1.0
f1-score	1.0	1.0	1.0	1.0	1.0
support	21.0	24.0	1.0	45.0	45.0

Fig-6.1 Accuracy of Random forest classifier

For just one split of data, the Random Forest classifier was able to achieve a 100% accuracy on both train and test data, but on 20 distinct sets of train and test split, the result was decreased to 95.90%. For the single split, the model was able to determine 24/24 falls, and 21/21 not falls, which could be seen in the confusion matrix graph in figure 6.2(b).

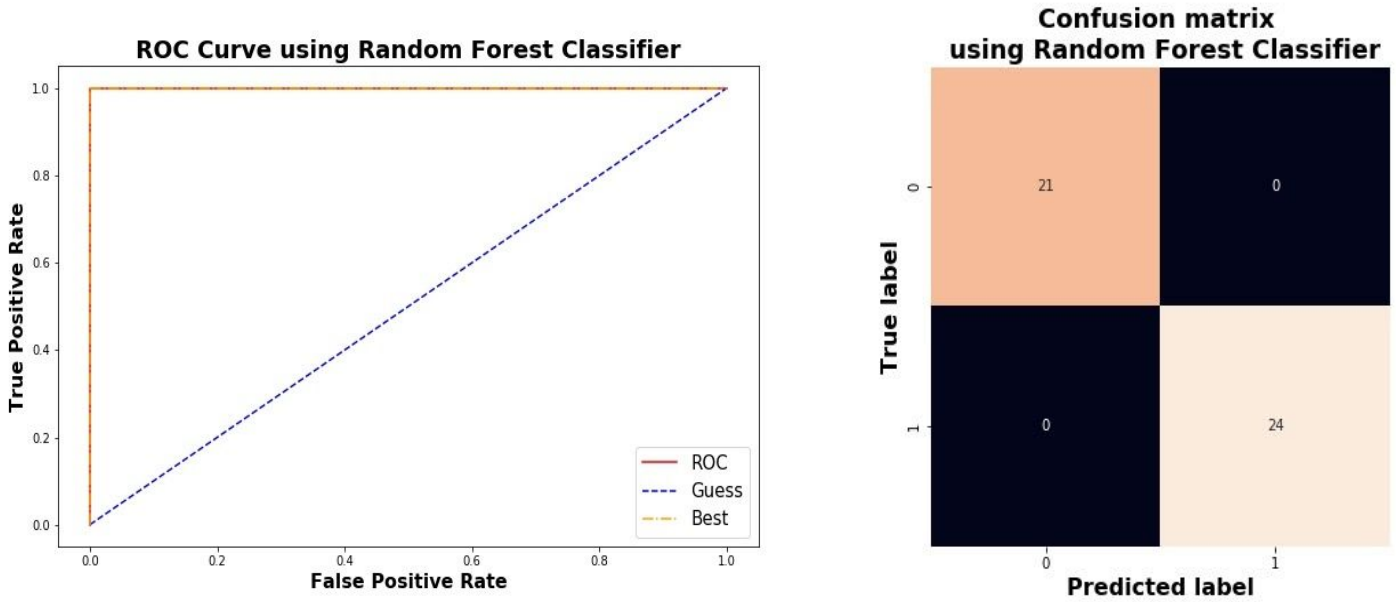


Fig-6.2 Random Forest Classifier (a) ROC CURVE (b) Confusion matrix

- AdaBoost Classifier

```
AdaBoost Classifier:
Train data Accuracy                          : 100.00%
Test data Accuracy                           : 100.00%
20-fold Accuracy                              : 97.71%
Receiver Operating Characteristic (ROC) Score : 1.0


Classification report :
              0     1  accuracy  macro avg  weighted avg
precision   1.0   1.0       1.0        1.0           1.0
recall      1.0   1.0       1.0        1.0           1.0
f1-score    1.0   1.0       1.0        1.0           1.0
support    20.0  25.0       1.0       45.0          45.0
```

Fig-6.3 Accuracy AdaBoost Classifier

For just one split of data, the AdaBoost Classifier was able to achieve a 100% accuracy on both train and test data, but on 20 distinct sets of train and test split, the result was decreased to 97.71%. For the single split, the model was able to determine 25/25 falls, and 20/20 not falls, which could be seen in the confusion matrix graph in figure 6.4(b).

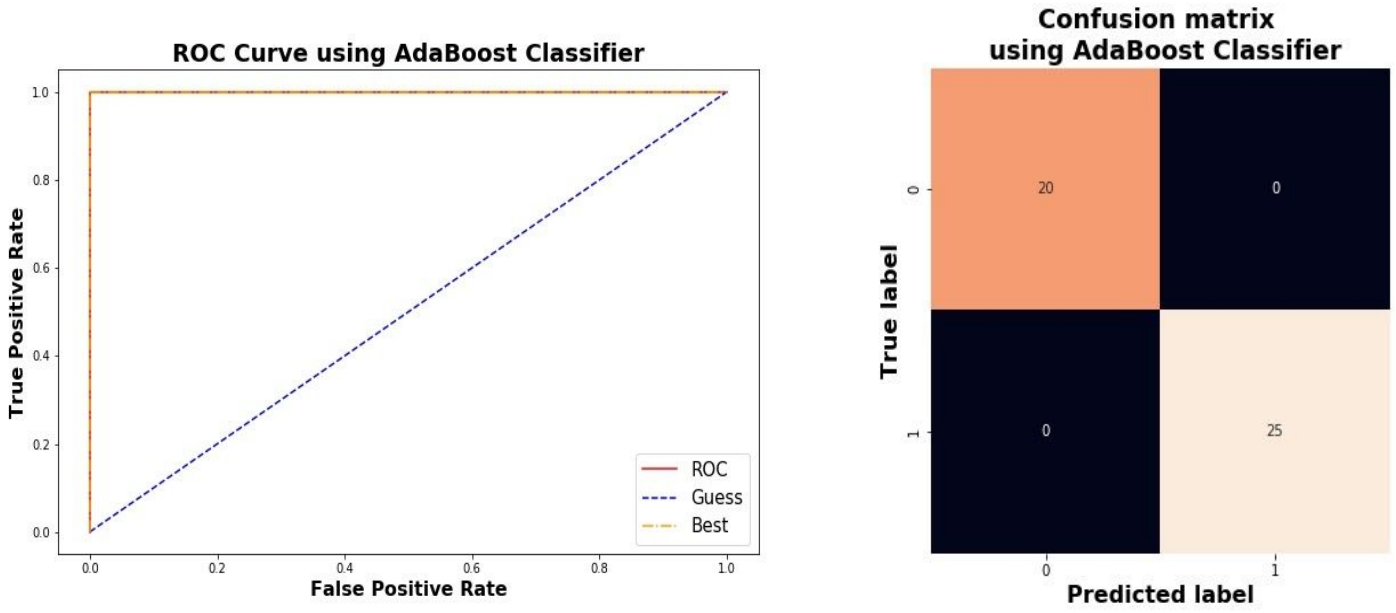


Fig-6.4 AdaBoost Classifier (a) ROC CURVE (b) Confusion matrix

- Multi-layer Perceptron classifier

```
Multi-layer Perceptron classifier  :
Train data Accuracy                              : 100.00%
Test data Accuracy                               : 100.00%
20-fold Accuracy                                  : 90.97%
Receiver Operating Characteristic (ROC) Score : 1.0


Classification report :
              0     1  accuracy  macro avg  weighted avg
precision   1.0   1.0       1.0        1.0           1.0
recall      1.0   1.0       1.0        1.0           1.0
f1-score    1.0   1.0       1.0        1.0           1.0
support    20.0  25.0       1.0       45.0          45.0
```

Fig-6.5 Accuracy of Multi-layer Perceptron classifier

For just one split of data, Multi-layer Perceptron was able to achieve a 100% accuracy on both train and test data, but on 20 distinct sets of train and test split, the result was decreased to 90.97%. For the single split, the model was able to determine 25/25 falls, and 20/20 not falls, which could be seen in the confusion matrix graph in figure 6.6(b).

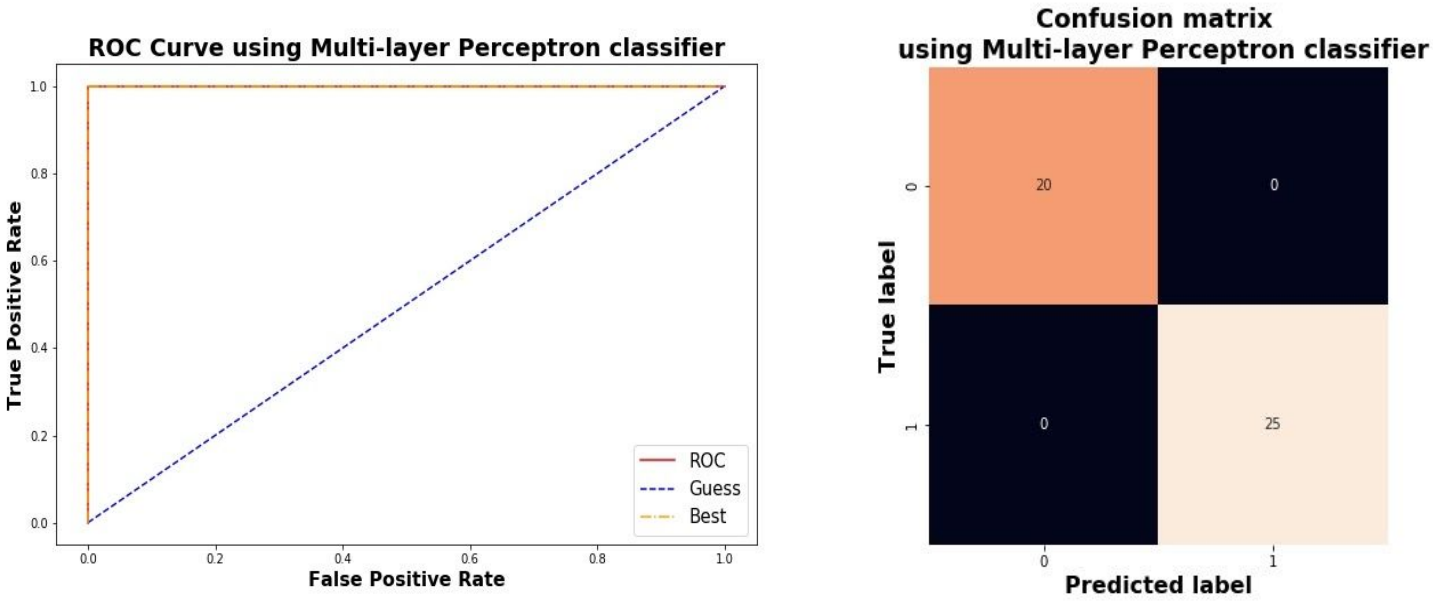


Fig-6.6 Multi layer perceptron (a) ROC CURVE (b) Confusion matrix

The testing scores and validation scores are summarized in Figure 6.7. Random Forest Classifier, AdaBoost Classifier, and Multi-layer Perceptron Classifier produced the best results. A lot of different hyperparameters were used to make the others model work, but, still, they produced suboptimal results with either overfitting or underfitting. Also, the 20-fold accuracy of the other models was a lot less than a single split. Out of the best three models, AdaBoost had the best accuracy, but all of the three models were taken for further analysis.

Trained models for Random Forest, AdaBoost, and MLP were saved using the pickle library. Saving the model allows the reuse of the model without going through the training process again, and a saved model could be used in further applications.

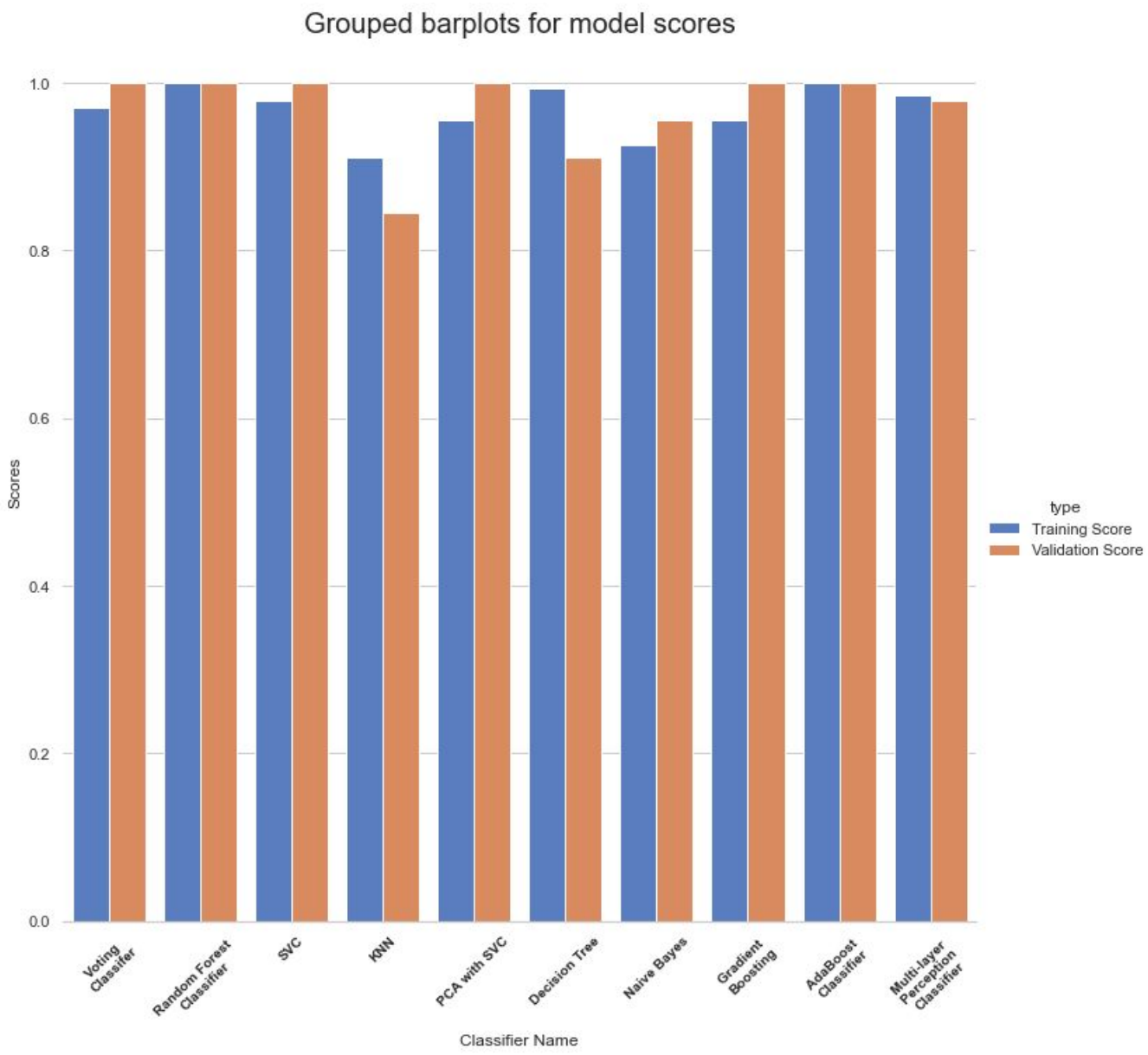


Fig-6.7 Train and a test score of each model

6.3 Predicting

The next step was to see how the models were gonna perform when predicting never-seen data. In total, fifteen new datasets were recorded using the same IOS app. All the steps required, i.e. cleaning,

transforming, loading the trained models, and predicting the results, are in a single python file. Figure: 6.8 shows the results of how our trained models predicted for each scenario.

Walk and fall predictions:

Random forest	AdaBoost	MLP
1	1	0
1	1	1
1	1	1

Walk and sit predictions:

Random forest	AdaBoost	MLP
0	0	0
0	0	0
0	0	0

Downstairs and sit predictions:

Random forest	AdaBoost	MLP
1	1	0
1	1	0
1	1	0

Run and fall predictions:

Random forest	AdaBoost	MLP
1	1	0
1	1	1
1	1	1

Freefall predictions:

Random forest	AdaBoost	MLP
1	1	1
1	1	1
1	1	1

Fig-6.8 predicted results for each scenario.

AdaBoost and Random forest classifier were able to predict perfect results for Walk Fall, Walk Sit, Run Fall, and Free Fall scenarios. But both of them predicted a fall for all of the three Down Sit scenarios. On the other hand, Multi-layer Perceptron did well when predicting Down Sit but missed two of the other scenarios.

Just to understand the behaviour and with what probability the models are producing the results for the Down Sit scenario, a predict_proba was used with each model. The results are in the figure:6.9. The results show that Multi-layer Perceptron was around 99% sure that those were not a fall, but the random forest was 67% - 83% sure for it to be a fall, and AdaBoost was around 67% sure for two readings.

Pridicted probability by Random forest :

Not-Fall	Fall
0.17	0.83
0.27	0.73
0.33	0.67

Pridicted probability by Ada Boost :

Not-Fall	Fall
0.055932664219750385	0.9440673357802496
0.32183885942881235	0.6781611405711877
0.32183885942881235	0.6781611405711877

Pridicted probability by MLP :

Not-Fall	Fall
0.9999986863898758	1.3136101242371043e-06
0.999017144449047	0.00098285555095297
0.9999986549742913	1.34502570874002e-06

Fig-6.9 predict_proba( ) on downstairs and sit

It was difficult to decide the best model, but they need to perform accurately for critical fall situations. Both of the Random forest and AdaBoost were able to predict the critical situations accurately. As AdaBoost had greater 20-fold accuracy, hence, it was chosen to be the most appropriate.

7. Results

A python file app.py was created, which would take a CSV file as input. The file is required to be similar to the one recorded by the “Sensor Data Recorder IOS application” and then would print two things. Firstly, if that was a fall or not and with what percentage of accuracy the model predicted the result.

```
That was a not a fall !
- With an accuracy of 95.0 %
```

Fig-7.1 app.py result for a walk and sit scenario

8. Limitations

For automatic fall detection and machine learning in general, ideally, it should have a lot more data to fine-tune the models. Due to time limitations, we were unable to do so. Instead, a limited five groups of scenarios were created and only collected around 45 data samples for each group. If time weren't a problem, more time would have been spent over-collecting data.

This whole project was contained inside Jupyter notebooks. These notebooks are useful for analysis, but they can't automatically perform fall detection. The ideal version of the project is to create an app that detects fall. Unfortunately, time was our enemy. After some considerations, it was decided to only focus on a smart portion of the whole picture.

It was realized that neural networks and deep learning are perfect for solving this problem. At some point, some time was spent on learning the concepts and tools, such as Tensorflow.js, and using them in the project. However, it was decided to drop the idea since it was not possible to learn a vast amount of knowledge within a limited amount of time. Instead, the machine learning models were used from what had been learned in the class.

9. Accomplishments

Joshua Leung

- Performed data collection
- Conducted statistical analysis on the transformed dataset using ANOVA
- Visualized the data with line charts and bar plots using matplotlib and seaborn
- Visualized the train and validation scores of each model using seaborn
- Trained and optimized various machine learning models to produce the best validation scores

Pruthviraj Patel

- Performed data collection
- Cleaned data and applied Butterworth filter for all scenarios to prepare for analysis
- Applied a custom function to generate transformed data files, making it easier to interpret and analyze data.
- Visualized the original data and filtered data using matplotlib and seaborn for comparison
- Trained various machine learning models to produce optimal training and validation scores

Harnoor Singh

- Performed data collection.
- Cleaned data using fast Fourier transform, visualized all the data before and after cleaning.
- Worked on transforming the data, tried different ways and added various columns using the forward attribute selection technique.
- Improved machine learning models with different hyperparameter selection using GridSearchCV.

- Worked on analyzing the machine learning results using different techniques such as ROC, Confusion matrix, Finding where and when the best models were failing.
- Worked on saving the model, then using it to predict the unseen data (predict.ipynb). Finally working on a python file which will take input and output the result (fall or not) and accuracy of result.