Machine Learning based Accessible Mobile App for Activity Recognition and Freezing of Gait Monitoring in Parkinson's Patients

Team 4, La Cueva High School, April 6, 2022

Team Members : Abitpal Gyawali, Aditya Koushik, Aiden Shoppel, Amandeep Prasankumar, Venkata Menta

Mentor : Jeremy Jensen

Table of Contents

Executive Summary

Parkinson's disease is a progressive neurological disorder that affects movement. It is caused by the death of cells in the brain that produce a chemical called dopamine, which helps regulate movement. An extremely evident symptom of Parkinson’s disease is the “Freezing of gait (FOG) .” This is when an individual’s feet gets “stuck” to the ground, making it difficult or impossible to take a step. Freezing of gait can be very distressing and significantly impact a person’s ability to move around and participate in activities. The current diagnosis treatment for detecting Parkinsons can be time consuming and involves the patient being given a series of questionnaires that provides insight to doctors regarding whether the patient has the disease. So in most cases the patient struggles with the disease before necessary medication and attention is given to the patient, and the problems remain to be persistent as medical professionals are unable to detect the extent of the disease as the diagnosis has no definitive test and is extremely vulnerable to misdiagnosis. In consideration of these disadvantages, this project sets an attempt to solve this problem by using a machine learning model built on a mobile app for activity recognition and freezing of gait detection enabled by the use of triaxial motors in present mobile phones. The problem solution is finding a more accessible, accurate and practical way of identifying Parkinson's disease by detecting freezing of gait via a smartphone app and machine learning. This app will also allow for an easy, hands-free way to keep track of how many FOG events occur per day which is a very useful diagnostic marker to identify progression or regression of disease. With support from the National Science Foundation and in collaboration with Rexa.Info, UC Irvine Machine Learning Repository released a dataset called the “Daphnet Freezing of Gait” in 2013. Using this published dataset, we trained an AdaBoost machine learning model on the dataset, where over 200 FOG episodes were recorded over the course of the experiment. The model learns based on a sliding window of FOG vs Non-FOG events, with over 500 statistical features (including fourier transform) being calculated over the windows. After successful training, an Activity Recognition model built with an XGBoost framework will use a similar sliding window approach to learn activities based on accelerometer data. The AdaBoost model with a Random Forest classifier baseline was trained and tested using a 5 fold cross validation and showed a 87% accuracy in distinguishing FOG windows vs Non - FOG Windows. The original Daphnet study achieved 73% sensitivity in predicting FOG events, while our model achieved 77%. Our XGBoost activity Recognition model scored >90% F1 scores in predicting activities such as walking, jogging, sitting, and standing.The next step taken was to integrate this study on the app, for the front end React Native(JS) CLI + Expo CLI was used to provide compatibility for iOS. In the backend Node JS, Express JS and Tensorflow JS were used. Finally, to obtain the accelerometer data from the phone, expo sensors were used. User testing of the app and further tuning of the models is still in progress.

Introduction

<u>Background: Parkinson's Disease, Freezing of Gait (FOG), Machine Learning Models, Fourier Transform</u>

- ***Parkinson's Disease***

1. *What is Parkinson's Disease?*

Parkinson's disease is a progressive neurological disorder that affects movement and is characterized by symptoms such as tremors, stiffness, and difficulty with movement and balance. It is caused by a loss of dopamine-producing cells in the brain. While there is no cure for Parkinson's disease, medication and other treatments such as deep brain stimulation can help manage symptoms and improve quality of life. Identifying early signs is important to allow for earlier intervention and support.

2. *What causes Parkinson's Disease?*

Parkinson's disease is caused by the death of dopamine-producing cells in a region of the brain called the substantia nigra. The exact cause of this cell death is not yet fully understood, but it is believed to be a combination of genetic and environmental factors. Research has identified several genetic mutations that may contribute to the development of Parkinson's disease, but these mutations are relatively rare and are thought to account for only a small percentage of cases. Environmental factors such as exposure to certain toxins, head injuries, and infections may also play a role in the development of Parkinson's disease, although more research is needed to fully understand these connections.There is also evidence to suggest that inflammation and oxidative stress may contribute to the development of Parkinson's disease. Inflammation can damage neurons and disrupt the normal functioning of the brain, while oxidative stress can damage cells and contribute to the death of dopamine-producing cells in the substantia nigra. Overall, Parkinson's disease is a complex condition with multiple factors contributing to its development. While the exact cause of Parkinson's disease remains unknown, ongoing research is working towards a better understanding of the condition and potential treatments.

3. *What are some common symptoms of Parkinson's Disease?*

Parkinson's disease is a neurological disorder that affects movement, and the symptoms can vary from person to person. However, some of the most common symptoms of Parkinson's disease include:

- Tremors: Uncontrollable shaking or tremors in one or more limbs, typically starting in one hand or arm.
- Bradykinesia: Slowness of movement, making everyday activities such as getting dressed or brushing teeth more difficult.
- Rigidity: Stiffness and resistance to movement in the limbs or trunk.
- Postural instability: Difficulty with balance and coordination, leading to falls.
- Freezing of gait: Brief episodes where the person's feet seem to get "stuck" to the ground, making it difficult to take a step.
- Changes in speech: Difficulty with speaking clearly or softly, slurring of words, or a monotone voice.
- Micrographia: Small, cramped handwriting.

Other symptoms may include fatigue, depression, anxiety, sleep disturbances, constipation, and loss of sense of smell. It's important to note that not all individuals with Parkinson's disease will experience all of these symptoms, and the severity and progression of symptoms can vary widely.

Freezing of Gait

1. *What is Freezing of Gait?*

Freezing of gait (FOG) is a common symptom of Parkinson's disease that affects a person's ability to initiate and continue walking. FOG is characterized by a sudden and brief inability to move the feet, making it feel like the person's feet are glued to the ground. It can occur when starting to walk, turning, or approaching an obstacle.

FOG is a complex symptom that can be caused by a combination of physical and cognitive factors. Physical factors include muscle weakness, poor balance, and rigidity, while cognitive factors can include anxiety, distraction, or hesitation. FOG can be particularly distressing for people with Parkinson's disease, as it can lead to falls and reduced mobility.

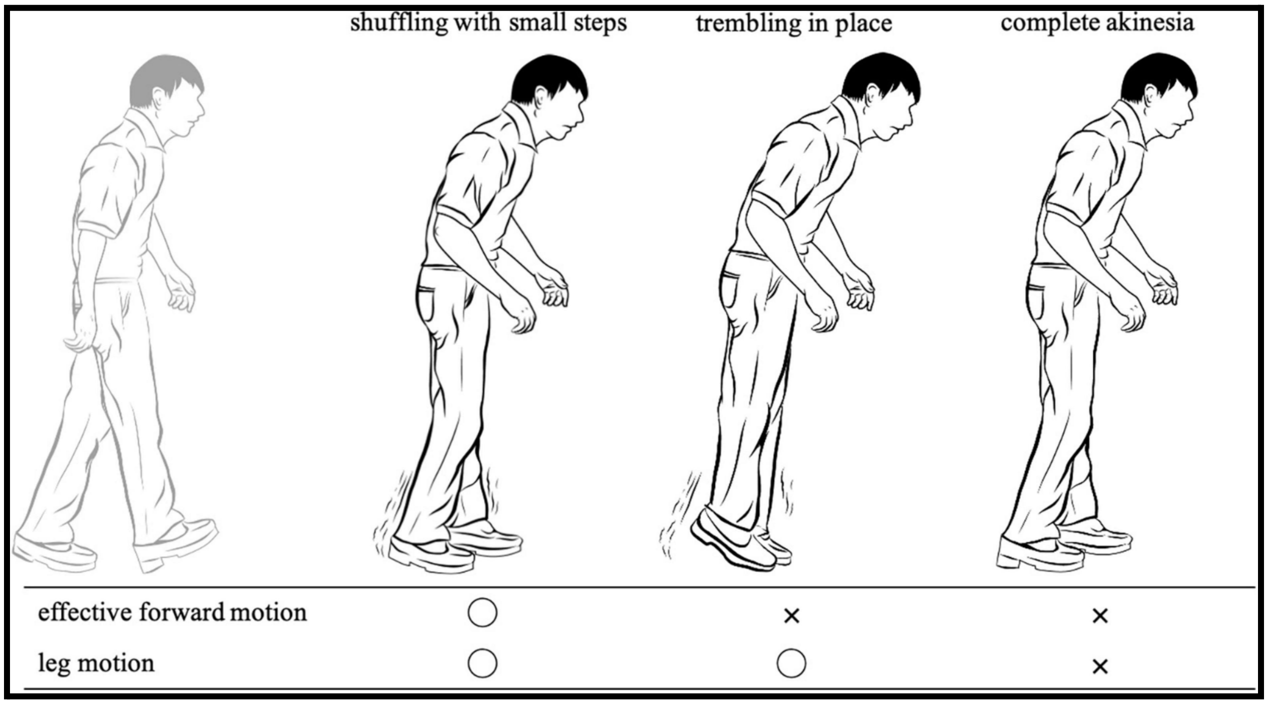


Figure 1

2. *How can Freezing of Gait be detected?*

Accelerometers are devices that can detect changes in movement and are commonly found on most smartphones. They can be used to detect freezing of gait (FOG) by measuring changes in movement patterns associated with FOG episodes. One way accelerometers can be used to detect FOG is by analyzing the gait pattern during normal walking and comparing it to the pattern during FOG episodes, which is performed by our machine learning model. During FOG, there may be a sudden decrease in the amplitude and frequency of lower limb movements. This reduction in movement can be detected by an accelerometer placed on the foot or leg.

- ***Ensemble Methods***

1. ***What are Ensemble methods?***

Ensemble methods are a set of machine learning techniques that combine multiple models to improve the accuracy and robustness of the predictions. Ensemble methods work by combining the predictions of several models, either through a voting system, weighted average or stacking, in order to produce a single more accurate prediction. Ensemble methods are especially useful in situations where the data is noisy or uncertain, as they can help to reduce the impact of errors or biases in individual models.Some of the most popular ensemble methods include bagging, boosting, and random forests. Bagging involves building multiple models on different subsets of the training data, and then averaging their predictions. Boosting, on the other hand, trains multiple models sequentially, with each model attempting to correct the errors of the previous model. Random forests combine the ideas of bagging and decision trees, by building an ensemble of decision trees on random subsets of the data. Ensemble methods have become increasingly popular in recent years, and are widely used in a variety of applications, including image and speech recognition, natural language processing, and recommendation systems.

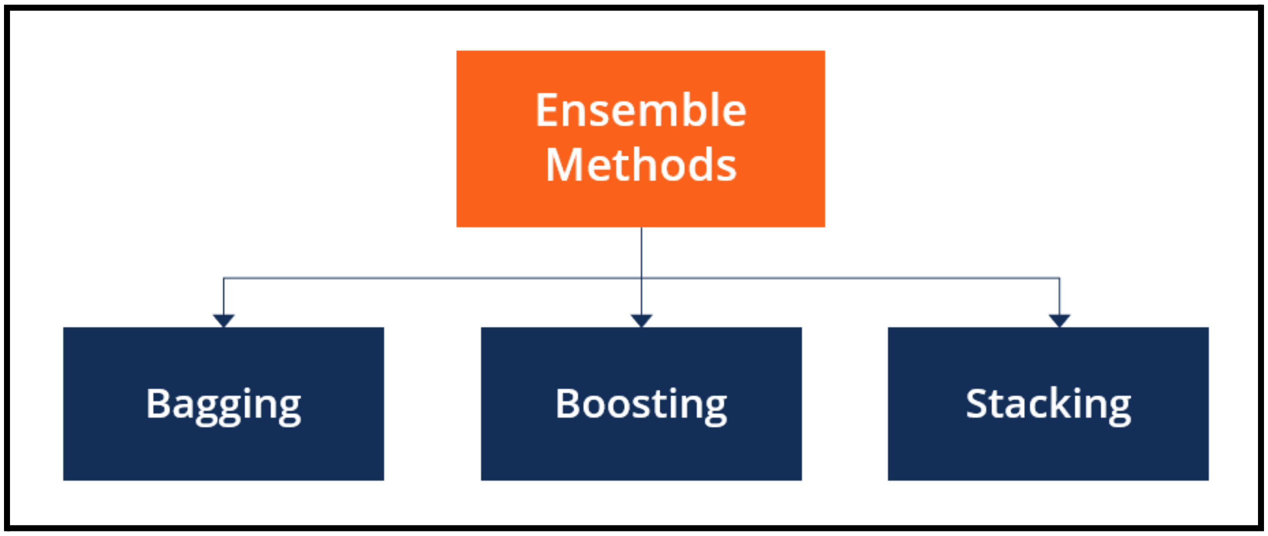


Figure 2

2. *What is ADABOOST?*

AdaBoost, short for Adaptive Boosting, is a machine learning ensemble algorithm that combines weak learners to create a strong learner. In AdaBoost, weak learners are simple models that perform slightly better than random guessing, such as decision trees with a small number of levels or simple regression models. AdaBoost iteratively trains these weak learners on the same dataset, with more emphasis on the samples that were misclassified by the previous model. This allows the subsequent models to focus more on the difficult samples that were misclassified, thereby improving the overall accuracy of the ensemble. During the training process, AdaBoost assigns weights to each sample in the dataset, with more weight given to the misclassified samples. In each iteration, the weak learner is trained on the weighted data, and a new weight is assigned to the model based on its accuracy. The final prediction of the ensemble is a weighted sum of the predictions of each individual model. AdaBoost is known for its high accuracy, robustness, and ability to handle complex datasets with high-dimensional feature spaces. It has been successfully applied to a wide range of applications, such as computer vision, natural language processing, and bioinformatics.

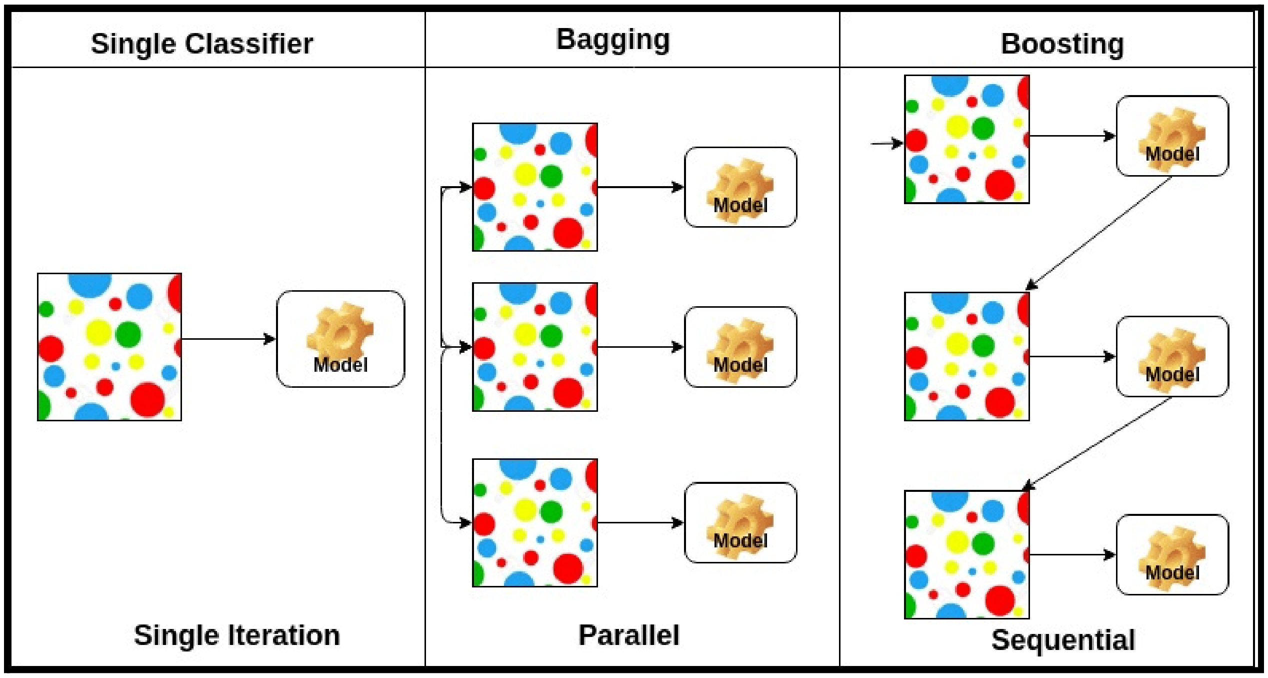


Figure 3

3. *What is XGBOOST?*

XGBoost stands for "Extreme Gradient Boosting," and it is a popular open-source machine learning algorithm for solving classification and regression problems. It is based on the gradient boosting framework and has gained popularity due to its speed, performance, and flexibility. The main idea behind XGBoost is to create a series of decision trees that can be used for making predictions on new data. Each decision tree is trained on a subset of the training data and the features are selected based on their importance in improving the performance of the model. The output of the decision trees is then combined in a way that maximizes the overall performance of the model.XGBoost has several key features that make it popular among data scientists and machine learning practitioners. These include:

- Speed: XGBoost is designed to be fast and efficient, making it suitable for large-scale datasets.
- Regularization: XGBoost has built-in regularization techniques such as L1 and L2 regularization to prevent overfitting and improve the generalization performance of the model.
- Handling missing values: XGBoost can automatically handle missing values in the data without the need for imputation techniques.
- Interpretability: XGBoost provides information on feature importance, which can help in understanding the underlying data patterns.

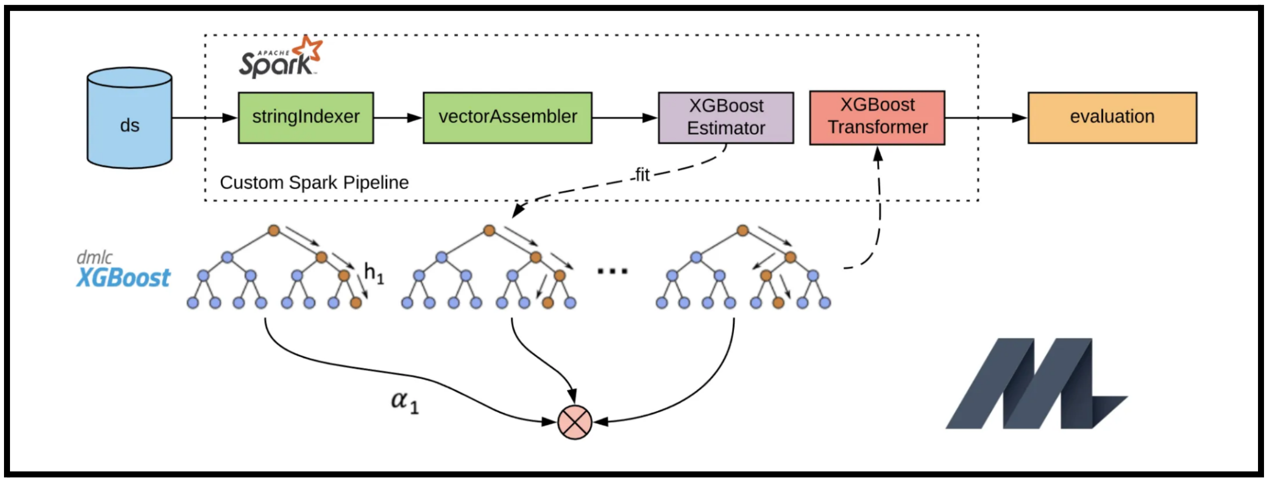


Figure 4

1. *What are Random Forests?*

Random Forests is a machine learning algorithm used for classification, regression, and other tasks. It is an ensemble learning method that combines multiple decision trees to create a more accurate and robust model. Random Forests works by building multiple decision trees on randomly selected subsets of the data and features. Each decision tree is trained on a random subset of the training data and a random subset of the features. The final prediction is then made by taking the majority vote (in classification) or the average (in regression) of the predictions of all the individual trees.

The main advantage of Random Forests is that it reduces overfitting by averaging the predictions of multiple trees. This makes it less sensitive to noise and outliers in the data, and less prone to overfitting compared to a single decision tree. Random Forests can also handle missing data and perform well with high-dimensional datasets. It can be used for both supervised and unsupervised learning tasks, such as classification, regression, clustering, and anomaly detection.

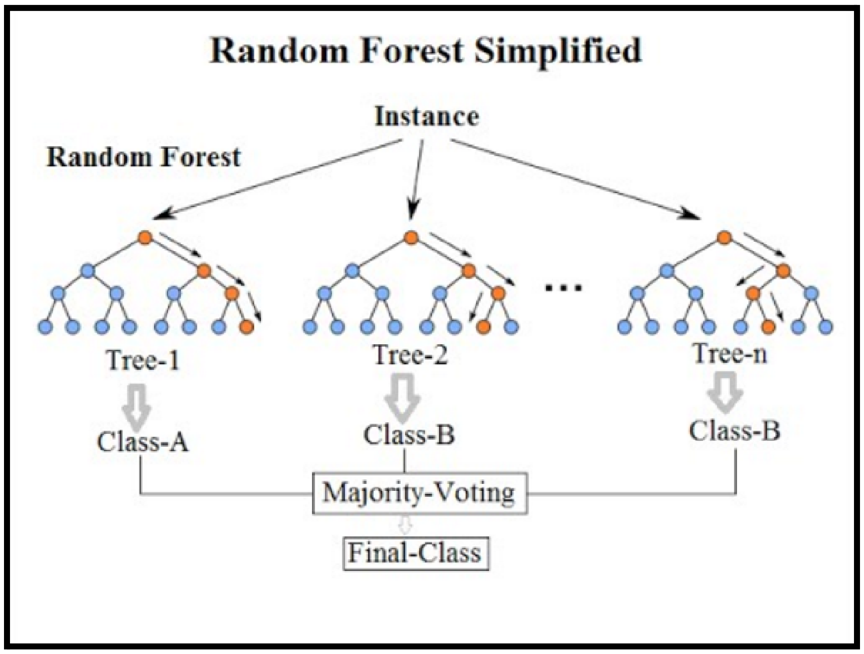


Figure 5

4. *What is Fourier Transform*

The Fourier transform is a mathematical technique that allows us to convert a signal or a function from its time or spatial domain to its frequency domain. In other words, it decomposes a complex signal or function into its constituent sinusoidal components. This transformation is named after Joseph Fourier, a French mathematician who discovered it in the early 19th century.

The Fourier transform is an important tool in time series forecasting because it can help identify the underlying periodicities and trends in a time series. By analyzing the frequency components of a time series using Fourier transform, we can identify patterns such as seasonality or cyclical behavior that are not immediately apparent from the raw data.Once the frequency components of a time series have been identified, various techniques can be used to model and forecast the time series based on those components. For example, one approach is to decompose the time series into its seasonal and trend components using techniques such as seasonal decomposition of time series (STL) or the Holt-Winters method. The Fourier transform can be used to extract the seasonal component of the time series, which can then be modeled and forecasted separately.

Another approach is to use Fourier analysis to identify the dominant frequencies in the time series and then use that information to build a forecasting model. This approach is often used in signal processing and can be applied to time series data as well. In summary, the Fourier transform is important in time series forecasting because it helps identify the underlying patterns and periodicities in the data, which can be used to build more accurate forecasting models. By using the Fourier transform as a tool for feature extraction and analysis, we can gain deeper insights into the structure of the time series and develop better forecasting models.

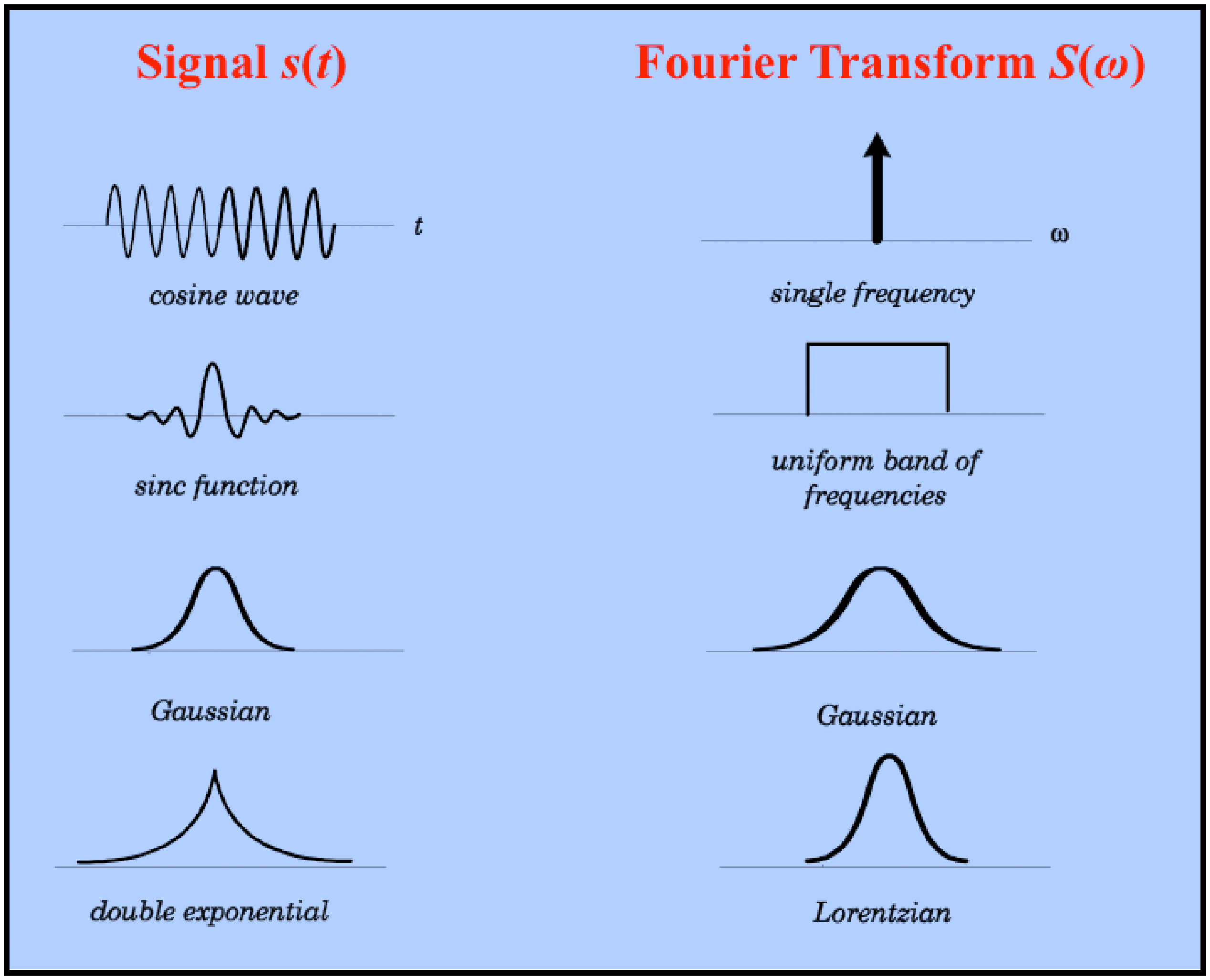


Figure 6 (Shows the following fourier transform or base sinusoid of the signal produced)

5. *What is the Fast Fourier Transform?*

The Fast Fourier Transform (FFT) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, which is a mathematical transformation that converts a signal from the time domain into the frequency domain. In simpler terms, the FFT is a way of analyzing a signal to determine the frequency components that make it up. For example, if you have an audio signal of a person speaking, the FFT can be used to determine the different frequencies of sound waves that make up that speech signal.

The FFT can be used in time series forecasting to identify and extract useful patterns from the data in the frequency domain. This can be particularly useful for identifying cyclic patterns or seasonality in the data.

Here are the steps to use the FFT in time series forecasting:

2. Preprocessing: The first step is to preprocess the time series data by removing any trends or seasonal components that can interfere with the frequency analysis. This can be done using techniques such as differencing or seasonal decomposition.
3. Compute the FFT: The next step is to compute the FFT of the preprocessed time series data. This will transform the data from the time domain to the frequency domain.
4. Identify the dominant frequencies: The FFT will produce a frequency spectrum that shows the various frequencies present in the data and their respective amplitudes. The analyst can identify the dominant frequencies by examining the frequency spectrum and selecting the frequencies with the highest amplitudes.
5. Forecasting: Once the dominant frequencies have been identified, they can be used to create a forecast by extrapolating the pattern into the future. This can be done by creating a sinusoidal function with the identified frequency and amplitude and projecting it into the future.
6. Validation: Finally, it is important to validate the forecast by comparing it to actual data. This can be done by comparing the forecasted values to the actual values and calculating the forecast error.

Overall, the FFT can be a powerful tool for time series forecasting as it allows analysts to identify and extract useful patterns from the data in the frequency domain.

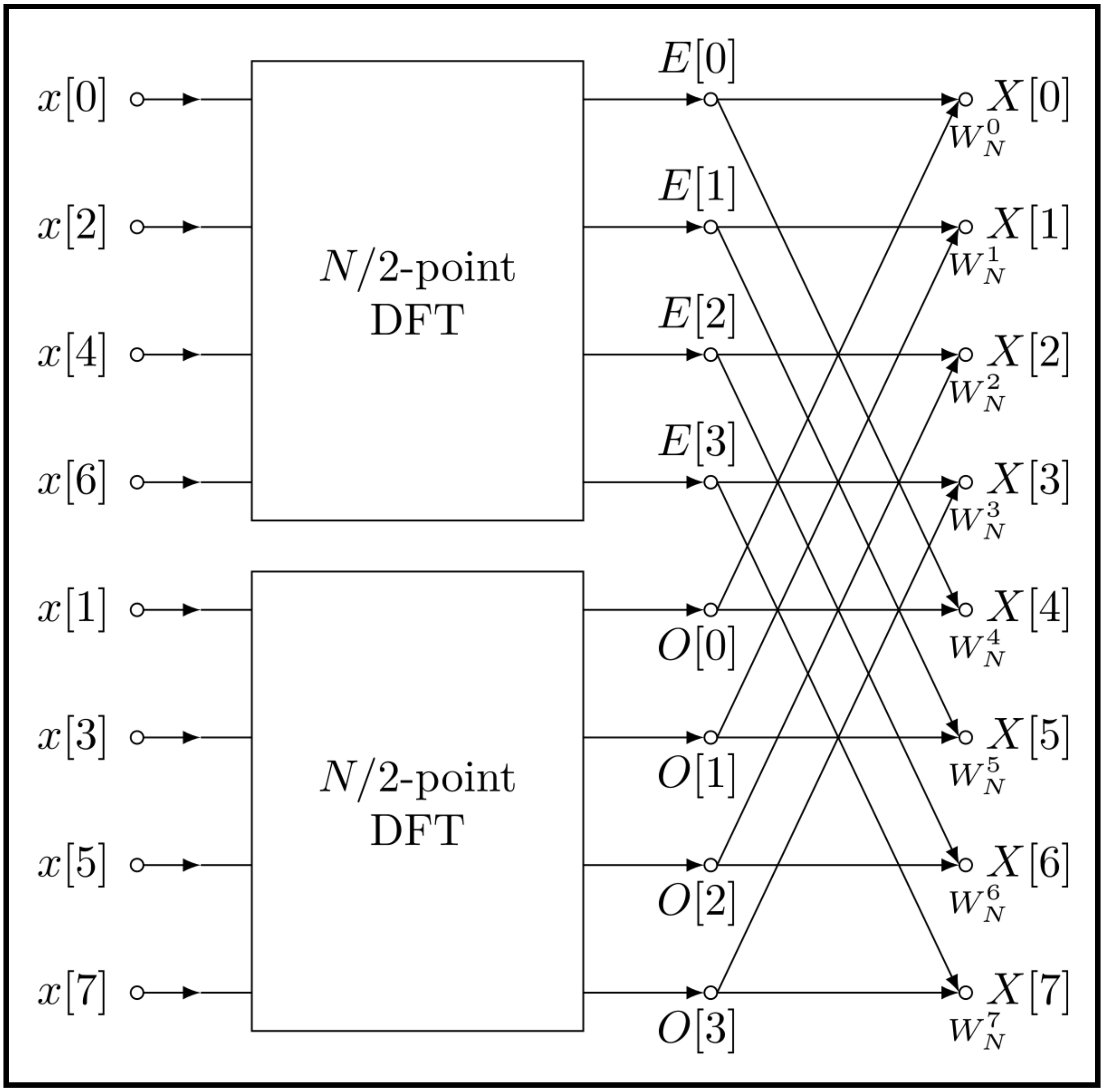


Figure 7 (The following illustration shows 8 data points being split into two groups of 4 points each. Each set goes through an algorithm that analyzes the data points in terms of frequency domains. which represent the signal in the frequency domain. The magnitude and phase of each data point is representative of the amplitude and phase of a corresponding frequency component in the signal. The DFT can then be effectively calculated using the FFT algorithm. The algorithm enables the use of the predictable nature of sinusoids and how they overlap to make the necessary computation)

Purpose

Finding a more accessible and practicable way of monitoring Parkinson's disease by detecting freezing of gait and user activities via a smartphone app and machine learning.

Materials

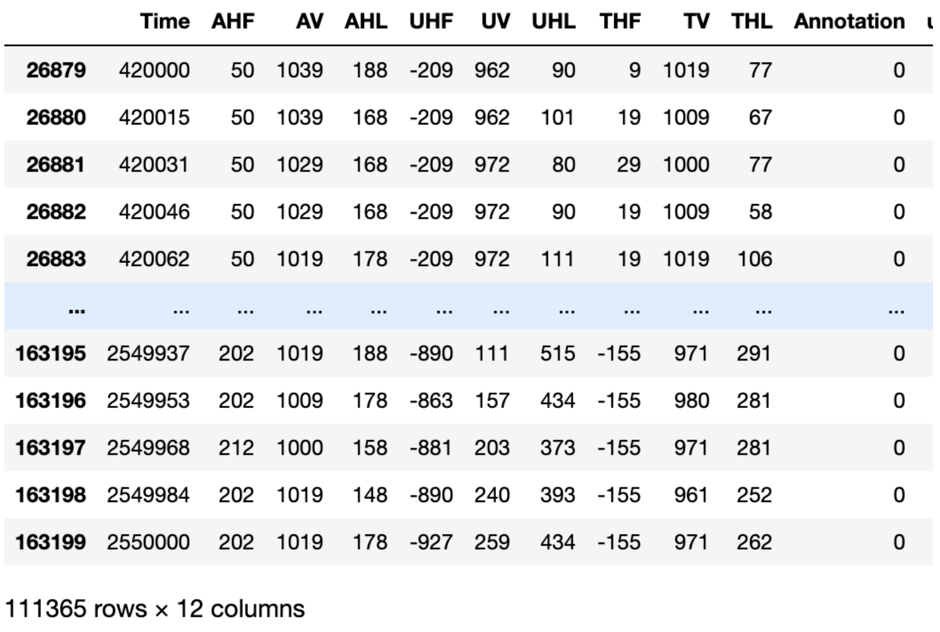
- Daphnet FOG dataset (https://archive.ics.uci.edu/ml/datasets/Daphnet+Freezing+of+Gait)
- WISDM Activity recognition dataset (https://www.cis.fordham.edu/wisdm/dataset.php)
- React Native (JS) CLI + Expo CLI
- Node JS
- Express JS
- Tensorflow JS
- Expo Sensors
- Python Flask
- Google Cloud Engine

Methods

- Creating the FOG model

Step 1: Loading the Dataset

The first step of the project was to train two models, the AdaBoost and XGBoost models, in being able to make accurate predictions from the Activity Recognition and Daphnet Freezing of Gait Dataset. From both datasets, upper thigh acceleration was recorded from three axes. The dataset used to train the model consisted of accelerometer data from subject 5 of the study sampled from three axes at the ankle, thigh, and trunk. Only the UHF (Upper leg thigh acceleration - horizontal forward acceleration [mg]), UV (Upper leg thigh acceleration - vertical [mg]) , and UHL (Upper leg thigh acceleration - horizontal lateral [mg]) acceleration data could be used to train the model, however, since the WISDM activity recognition dataset consisted of values only measured from the thighs.



	Time	AHF	AV	AHL	UHF	UV	UHL	THF	TV	THL	Annotation
26879	420000	50	1039	188	-209	962	90	9	1019	77	0
26880	420015	50	1039	168	-209	962	101	19	1009	67	0
26881	420031	50	1029	168	-209	972	80	29	1000	77	0
26882	420046	50	1029	168	-209	972	90	19	1009	58	0
26883	420062	50	1019	178	-209	972	111	19	1019	106	0
...	...	...	...	...	...	...	...	...	...	...	...
163195	2549937	202	1019	188	-890	111	515	-155	971	291	0
163196	2549953	202	1009	178	-863	157	434	-155	980	281	0
163197	2549968	212	1000	158	-881	203	373	-155	971	281	0
163198	2549984	202	1019	148	-890	240	393	-155	961	252	0
163199	2550000	202	1019	178	-927	259	434	-155	971	262	0

111365 rows × 12 columns

Figure 8: Daphnet dataset

The dataset also consisted of a Time column which was measured in milliseconds (ms) where acceleration values were recorded every 15 ms over the course of 36 minutes (duration of the experiment). The final and most important column is the "Annotation" column, where a 0 indicates that a given acceleration reading did not occur during a FOG event (No-FOG) and a 1 indicates that the acceleration reading occurred during a FOG event.

Step 2: Training the model

After the FOG dataset was loaded in, feature calculation was performed through the tsfresh python library, which automatically uses a sliding window through the dataset to compute over 1000 features. In order to prevent high dimensional input data, only a certain number of the original features could be selected for training the model. Most Calculated features showed a negligible correlation with the occurrence of FOG events, so $r > 0.1$ was used as the filter to acquire slightly correlated features with FOG activity. In total, 593 features were calculated and would be used to train the model.

Because the data was a time series dataset in nature, classical machine learning techniques could not be used to predict groups of collective anomalies or FOG events. Because of this, a sliding window (Figure 9) with a size of 50 data points (50 acceleration values) would scan through the dataset and the tsfresh library would compute features over each window of data. Examples of a few of the features calculated are sum of acceleration values, mean acceleration in the window, median acceleration, standard deviation of acceleration, kurtosis, etc. These features would be computed for UHF, UV, and UHL accelerations (x,y, and z axis). A window size of 50 was decided since it resulted in highest model accuracy when compared to other window sizes below 66. A window size of 66 would be a window of approximately 1 second of acceleration data (66*15), which would also mean that there would be a 1 second latency period between model prediction and live data collection. 1 second or 66 rows was decided as the max window size (a larger number would result in a greater latency), and 50 rows was the smallest number that seemed to show the best model performance. Another variable that had to be decided was if a window would be labeled as a FOG or Non-FOG event. Based on extensive testing and experimenting, it was found that a threshold of 0.4 seemed to show the best model performance (If 40% of annotations in the window are a 1, then the entire window would be classified as a FOG window). This variable has the greatest influence on model prediction, so further tuning of this parameter upon testing of the app is a future goal.

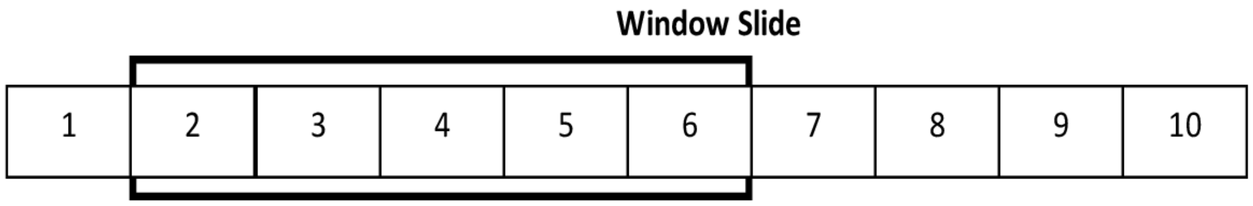


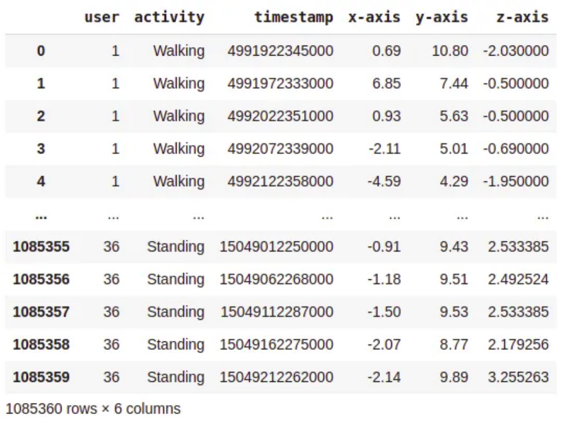
Figure 9: Example of sliding window

After all sliding windows were compressed into a single row or data value, the final dataset ultimately had 1356 rows where each row represents a window of 50 rows of the original data. After the scikit learn min-max scaler was applied to normalize each column of the data between values of 0 and 1, an AdaBoost model with a random forest baseline classifier was trained on the data. The model would be validated through a 5-fold cross validation and a LOOCV (Leave one out cross validation) as described in the results.

- **Creating the Activity Recognition Model**

Step 1: Loading the Dataset

The WISDM activity recognition dataset was loaded into the jupyter notebook file with the pandas module. The dataset consisted of ~1,000,000 rows of triaxial thigh accelerometer data from 36 users as shown by Figure 10.

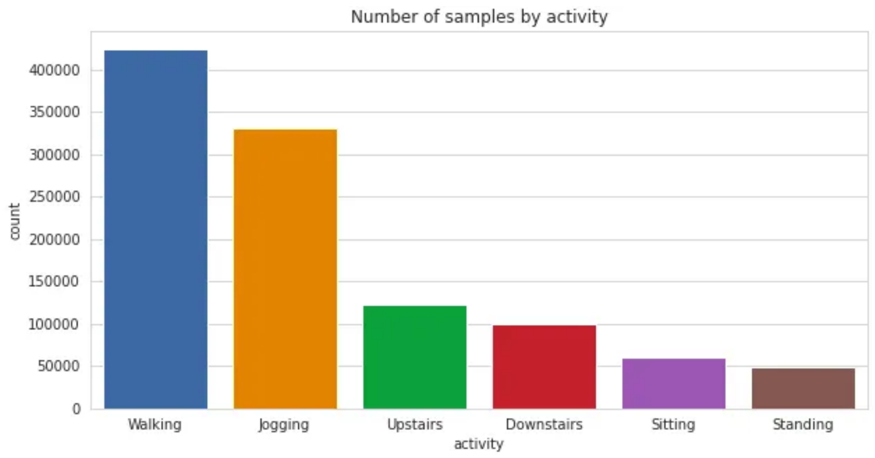


	user	activity	timestamp	x-axis	y-axis	z-axis
0	1	Walking	4991922345000	0.69	10.80	-2.030000
1	1	Walking	4991972333000	6.85	7.44	-0.500000
2	1	Walking	4992022351000	0.93	5.63	-0.500000
3	1	Walking	4992072339000	-2.11	5.01	-0.690000
4	1	Walking	4992122358000	-4.59	4.29	-1.950000
...	...	...	...	...	...	...
1085355	36	Standing	15049012250000	-0.91	9.43	2.533385
1085356	36	Standing	15049062268000	-1.18	9.51	2.492524
1085357	36	Standing	15049112287000	-1.50	9.53	2.533385
1085358	36	Standing	15049162275000	-2.07	8.77	2.179256
1085359	36	Standing	15049212262000	-2.14	9.89	3.255263

1085360 rows × 6 columns

Figure 10

The "activity" column consisted of annotation for 6 possible activities the user was engaging in: Walking, Jogging, Sitting, Standing, going upstairs and going downstairs (Figure 11 and 12). Because the goal of creating the activity recognition model was to determine when the FOG model should be activated, the model needed to be able to accurately predict when a user is walking/jogging vs sitting/standing.



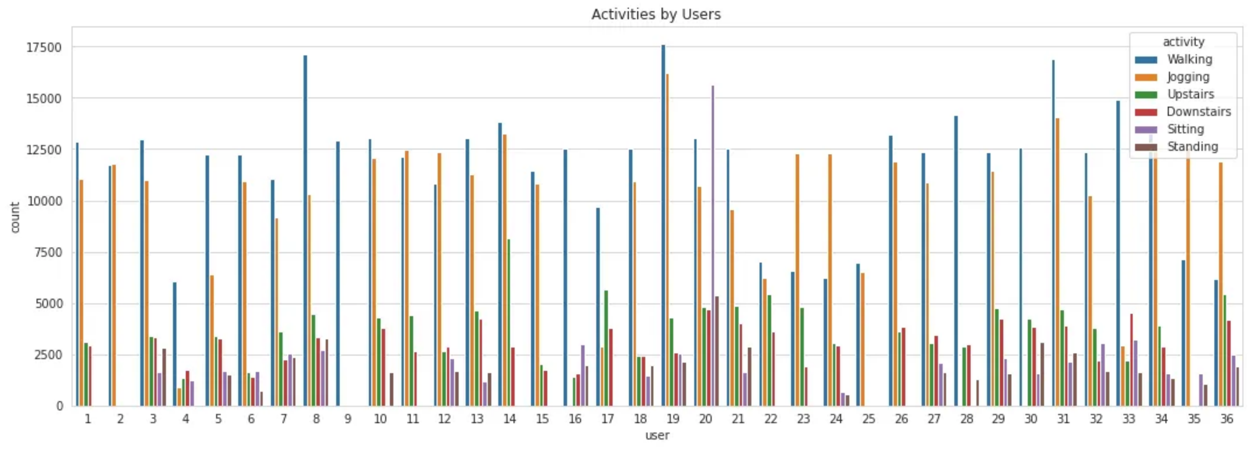


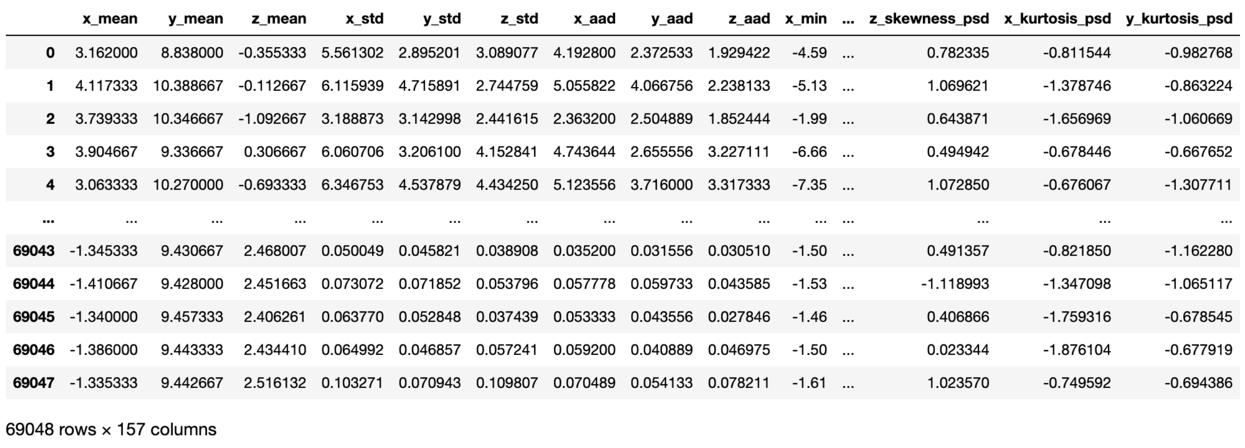
Figure 11 and 12

Step 2: Training the model

The process for training the model was similar to the FOG model. Because this dataset was sampled at 50 ms, a 15 row window size had to be created in order to produce the same prediction time (0.75 seconds) for both FOG model and activity recognition model (FOG: 50

rows * 15 ms vs Activity Recognition: 15 rows * 50 ms). The process of feature engineering was slightly different from the FOG model.

First, 18 simple statistical features were calculated for each axis on the 15-row sliding window as the entire dataset was scanned (18*3 = 54 features). These statistical features condensed accelerometer data from 15 rows into 1 value for each of the three axes of acceleration. The calculated features were mean acceleration, standard deviation, average absolute deviation, minimum acceleration, maximum, difference of maximum and minimum values, median, median absolute deviation, interquartile range, negative values count, positive values count, number of values above mean, number of peaks, skewness, kurtosis, energy, average resultant acceleration, and signal magnitude area. Finally, the fast fourier transform and power spectral density functions were applied to each window and all above features were recalculated from the new arrays. After the feature engineering, Figure 13 shows the final dataset before training the model, where each row in the below dataset represents a window of 15 rows in the original dataset. In total 156 features (excluding label column) were engineered from the original triaxial acceleration values.



	x_mean	y_mean	z_mean	x_std	y_std	z_std	x_aad	y_aad	z_aad	x_min	...	z_skewness_psd	x_kurtosis_psd	y_kurtosis_psd
0	3.162000	8.838000	-0.355333	5.561302	2.895201	3.089077	4.192800	2.372533	1.929422	-4.59	...	0.782335	-0.811544	-0.982768
1	4.117333	10.388667	-0.112667	6.115939	4.715891	2.744759	5.055822	4.066756	2.238133	-5.13	...	1.069621	-1.378746	-0.863224
2	3.739333	10.346667	-1.092667	3.188873	3.142998	2.441615	2.363200	2.504889	1.852444	-1.99	...	0.643871	-1.656969	-1.060669
3	3.904667	9.336667	0.306667	6.060706	3.206100	4.152841	4.743644	2.655556	3.227111	-6.66	...	0.494942	-0.678446	-0.667652
4	3.063333	10.270000	-0.693333	6.346753	4.537879	4.434250	5.123556	3.716000	3.317333	-7.35	...	1.072850	-0.676067	-1.307711
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69043	-1.345333	9.430667	2.468007	0.050049	0.045821	0.038908	0.035200	0.031556	0.030510	-1.50	...	0.491357	-0.821850	-1.162280
69044	-1.410667	9.428000	2.451663	0.073072	0.071852	0.053796	0.057778	0.059733	0.043585	-1.53	...	-1.118993	-1.347098	-1.065117
69045	-1.340000	9.457333	2.406261	0.063770	0.052848	0.037439	0.053333	0.043556	0.027846	-1.46	...	0.406866	-1.759316	-0.678545
69046	-1.386000	9.443333	2.434410	0.064992	0.046857	0.057241	0.059200	0.040889	0.046975	-1.50	...	0.023344	-1.876104	-0.677919
69047	-1.335333	9.442667	2.516132	0.103271	0.070943	0.109807	0.070489	0.054133	0.078211	-1.61	...	1.023570	-0.749592	-0.694386

69048 rows × 157 columns

Figure 13

The above dataset was normalized with the min max scaler and was used to train an XGBoost model to make multiclass predictions of a user's activity based on the 156 provided features. The model's accuracy was evaluated through a 5-fold cross validation test.

App Creation

Terminology

Front-end: Frontend is the portion of the app that is shown to the user. It consists of the User Interface (UI) and must be able to handle user inputs; front-end handles all the interactions with the user.

Back-end: Backend of an app is the portion of the app that handles data computation, data storage, and any other logical processes that allow the app to operate. In other words, the back-end handles the logic of the application, and does not interact with the user.

API (Application Programming Interface): APIs are essentially the connection between the front-end and the back-end of the application. It allows these two components to communicate with one-another through a series of requests and responses.

Objective of the Application

The table below demonstrates what each component of the app must be able to do:

Front-end	Back-end	API
- Be able to read and collect iPhone accelerometer data - Be able to send the collected accelerometer data to the back-end - Have a consistent UI that allows the user to navigate the app smoothly - Be able to interact with the back-end	- Be able to receive accelerometer data from the front-end through an API - Be able to feed in the received accelerometer data into a machine learning model and obtain a prediction - Be able to send the model prediction to the front-end through	- Create a stable connection between the front-end and the back-end - Be able to send accelerometer values from the front-end to the back-end - Be able to send predictions from the back-end to the front-end

through an API	an API	

Step 1: Front-end Prototype - Choosing the Software

The first step of creating a front-end for the application is to decide which software to use to make the application. There are several main questions that go into deciding the software:

- What platform will the application run on? Android? iOS?
- Will the application be available on all major platforms?
- What kind of tools are present in the application?
- What kind of tools does the application need access to?

We wanted the application to be available on all major platforms (most notably iOS and Android). In addition to that, the application must be able to read and store accelerometer values of the cell-phone. React Native, a software built by facebook for app development, fulfilled both of these major requirements. In addition to fulfilling these two requirements, our team also had prior experience with React JS (the Javascript library that React Native is built on), allowing us to build the application without having to learn a new language.

Collecting accelerometer values of the cell phone

In order to collect the accelerometer values of the user's phone, we used a React Native library called Expo Sensors. However, Expo Sensors could not be used without implementing the app using Expo CLI. Therefore, our initial prototype of the app was built using Expo CLI. However, in order for our application to fully function, we must be able to send this accelerometer data to our backend through an API. Since we used a local computer to run the backend server (through local host) during development, our server was hosted under HTTP protocol, rather than a more secure HTTPS protocol. However, due to Apple's security protocols, requests to insecure sites (such as those not protected by HTTPS) are blocked. This can be avoided through changing the application's metadata. However, this change of metadata could not be done through Expo CLI due to its high level nature. This problem was fixed through creating a project through the command line command "create-react-native-app", which allowed projects to incorporate both React Native CLI and Expo CLI. This essentially allowed us to maintain access to the phone's accelerometer data while enabling control over the application's metadata.

The code below creates an exception to Apple's security protocol, allowing the application to connect to my computer locally.

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
    <key>NSExceptionDomains</key>
    <dict>
        <key>{Domain}</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <true/>
        </dict>
    </dict>
</dict>
```

Accelerometer values are collected as follows:

- Accelerometer values are collected every 15 milliseconds
- At each instance of collection, acceleration in the x-axis, y-axis, and z-axis are stored in their respective list

- After a certain amount of accelerometer data is collected, the arrays of x-axis acceleration, y-axis acceleration, and z-axis acceleration are sent to the backend through a POST request, and the array is cleared

This overall process is shown in the pseudo code below:

```
x_list = []
y_list = []
z_list = []

data_length = {model’s ideal length of data}

//app is running
//DeviceMotion handles the accelerometer value events

DeviceMotion.addListener((accelerometer_data)=>{
     x_list.push(accelerometer_data.x)
     y_list.push(accelerometer_data.y)
     z_list.push(accelerometer_data.z)

     if x_list.length == data_length
          POST Request —> body: [x_list, y_list, z_list]

})
```

This allows for the prototype of the front-end with essentially all the necessities of the application.

Step 2: Back-end Prototype - Choosing the Software

As aforementioned, the backend of the application will handle the application's interactions with the machine learning model. This includes preparing the accelerometer data for model input, which mainly consists of computing the features of the accelerometer data (these features are the inputs to the models). Both the activity recognition model and the FOG detection models utilize complex statistical methods to compute the features. Therefore, it is in our best interest to use a language such as Python, that has the tools necessary to compute these features. In addition to that, the original model was trained on features computed using Python libraries, so in order to minimize the chances of errors in the code, we decided to use Flask (a Python Back-end library) to create our backend. This would allow us to mimic the original method of feature engineering.

Overall Process (Backend) :

The overall process for the backend is as follows:

1) Input arrays (x-list, y-list, z-list) are converted from JSON to a python array
2) The features are computed for the array (for both the activity recognition model and the FOG detection model)
3) Predictions are made by the model
4) These predictions are sent to the frontend as a response to the original POST request

Step 3: Testing

We tested the application on two main components: prediction when the user is still and when the user is walking. The inefficiencies of setting up a controlled experiment to test the application's detection of upstairs (walking upstairs) and downstairs (walking downstairs) discouraged us from testing these two types of predictions. In order to test still and walking, we recorded the accuracy of the application's predictions when the user is still (sitting and standing still) for three minutes, and accuracy of the predictions when the user is walking continuously for three minutes. We perform several trials of these tests (2~3). Through these testing periods, the iPhone (our testing device) was put inside a pocket, located on the upper thigh. These tests were conducted in a house to make sure the conditions of testing were realistic.

Window Size	15	50
Accuracy - Still	98%	100%
Accuracy - Walking	79%	Data Collection in Progress

Points of Errors → Errors mostly occurred during these conditions:

- Fast paced walking – errors seldom occurred when the user was walking faster, in which the model would predict the user was walking. This most likely occurred from the fact that the training data was taken from older patients.
- Turning – although errors during moderate turns did not occur often, sharp turns would often cause the model to predict the user was jogging

Although the data collection is in progress for measuring the accuracy of walk predictions for the window size of 50, from what we have gathered so far, the accuracy has been between mostly around 85%, showing a significant increase from the 79% with the window size of 15.

Google Cloud App Engine

After enough tests have been performed and the model has been tuned, the app will be run on the google cloud app engine. This will allow for the application to be easily scalable, and will also allow the application to bypass Apple's security guidelines regarding secure domain connections. Figure 14 shows the predicted logic of the application after development.

Progress

So far, the application can do the following:

Front-end	Back-end	API

- Be able to read and collect iPhone accelerometer data - Be able to send the collected accelerometer data to the back-end - Be able to interact with the back-end through an API	- Be able to receive accelerometer data from the front-end through an API - Be able to feed in the received accelerometer data into a machine learning model and obtain a prediction - Be able to send the model prediction to the front-end through an API	- Create a stable connection between the front-end and the back-end - Be able to send accelerometer values from the front-end to the back-end - Be able to send predictions from the back-end to the front-end

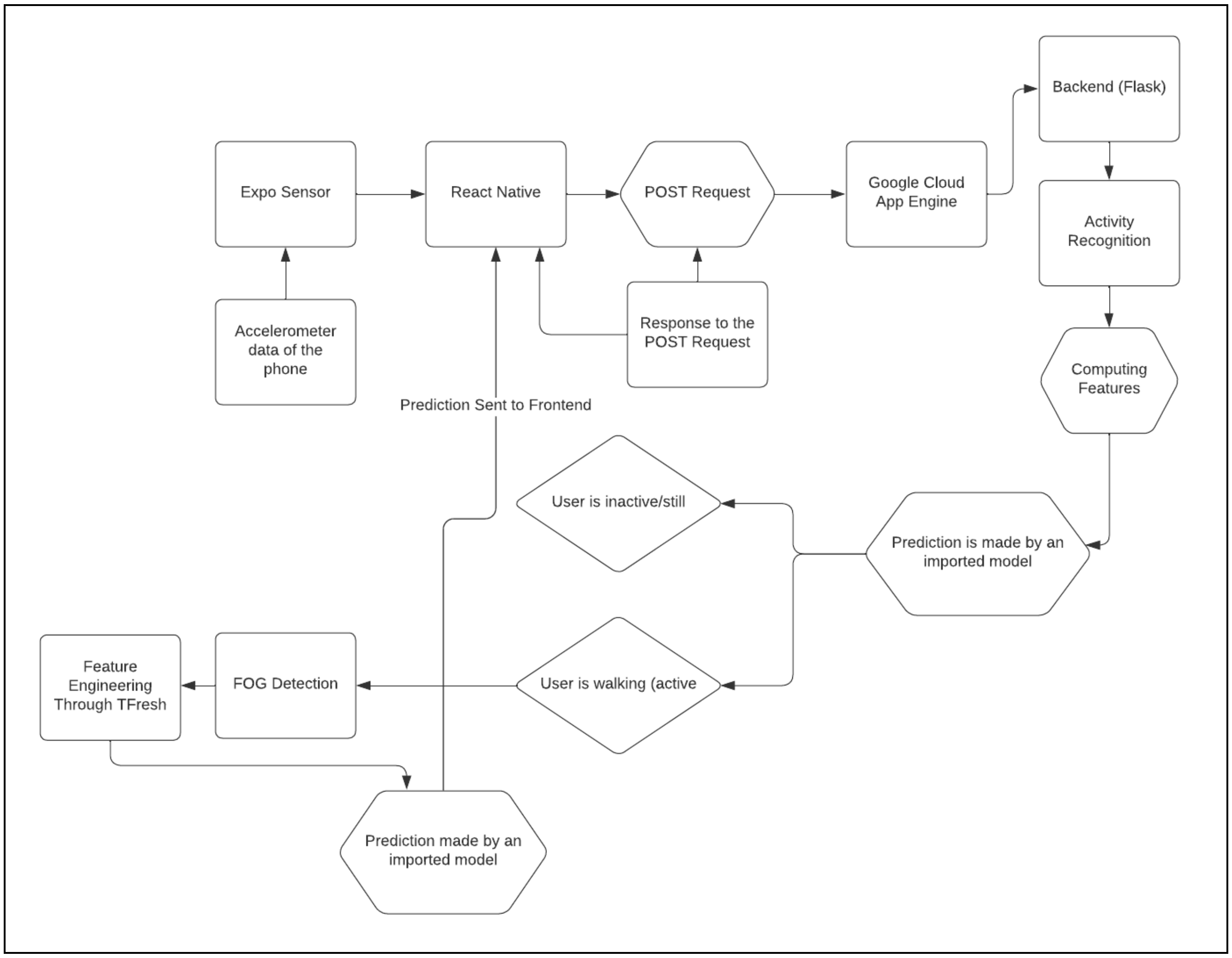
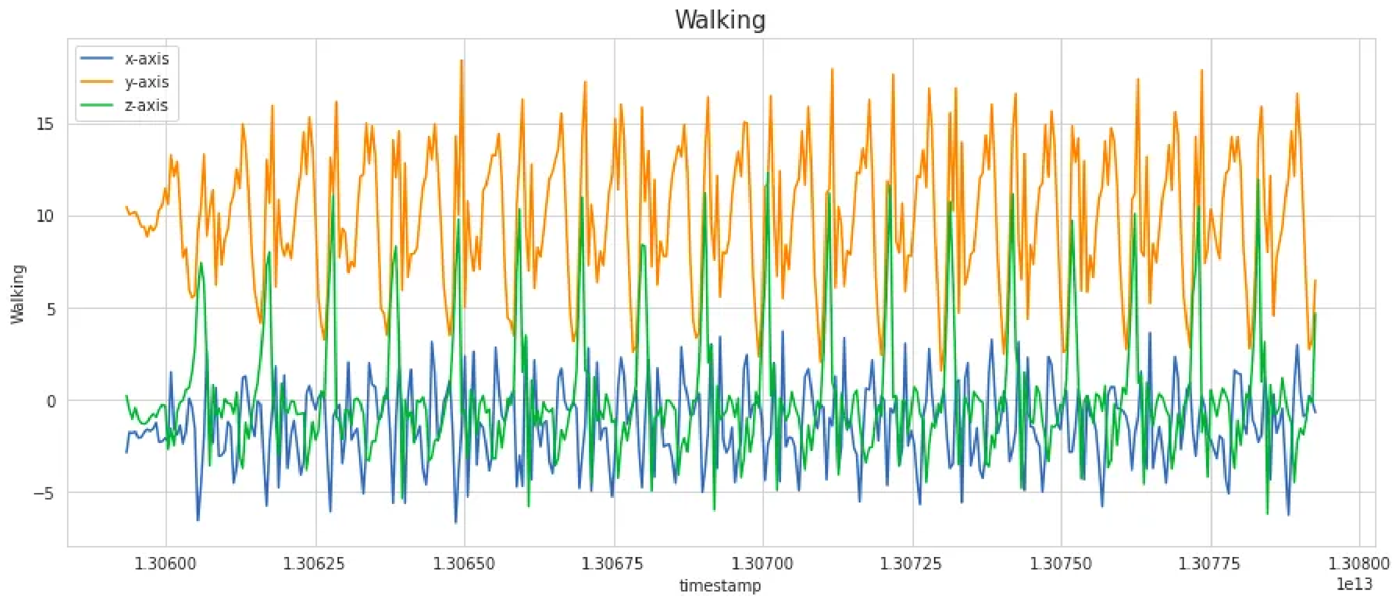
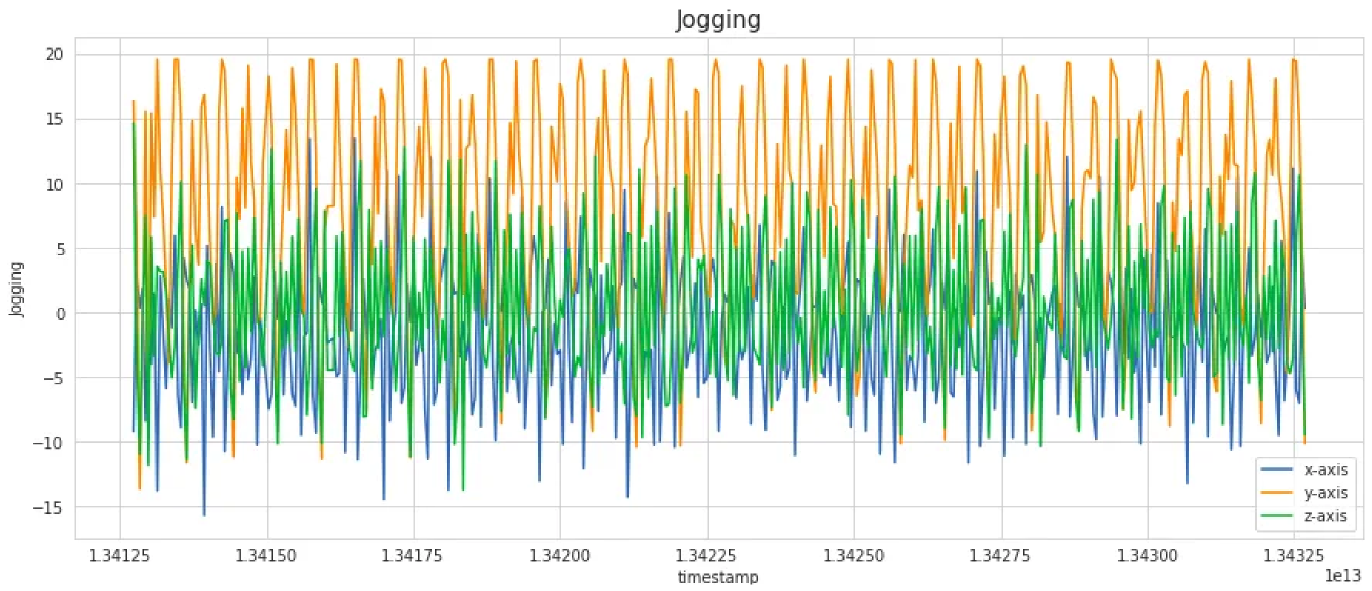

Backend (Flask)
Expo Sensor
React Native
POST Request
Google Cloud App Engine
Activity Recognition
Accelerometer data of the phone
Response to the POST Request
Computing Features
Prediction Sent to Frontend
User is inactive/still
Prediction is made by an imported model
Feature Engineering Through TFresh
FOG Detection
User is walking (active
Prediction made by an imported model

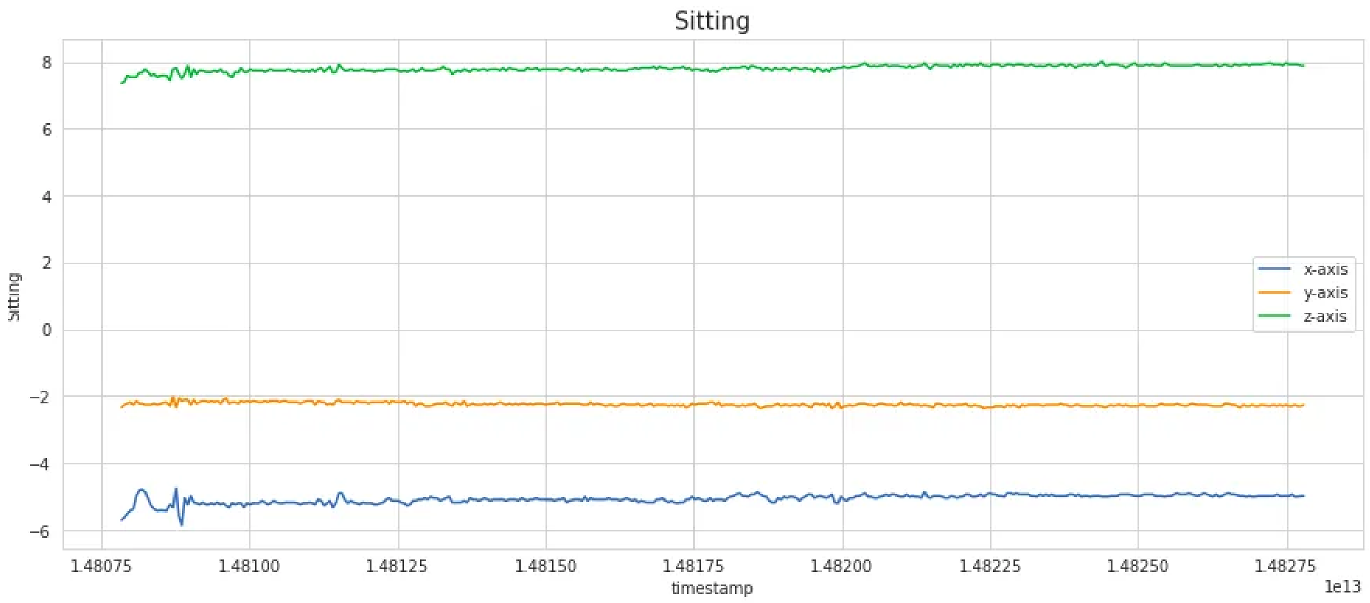
Figure 14: Overview of App - Application Flowchart

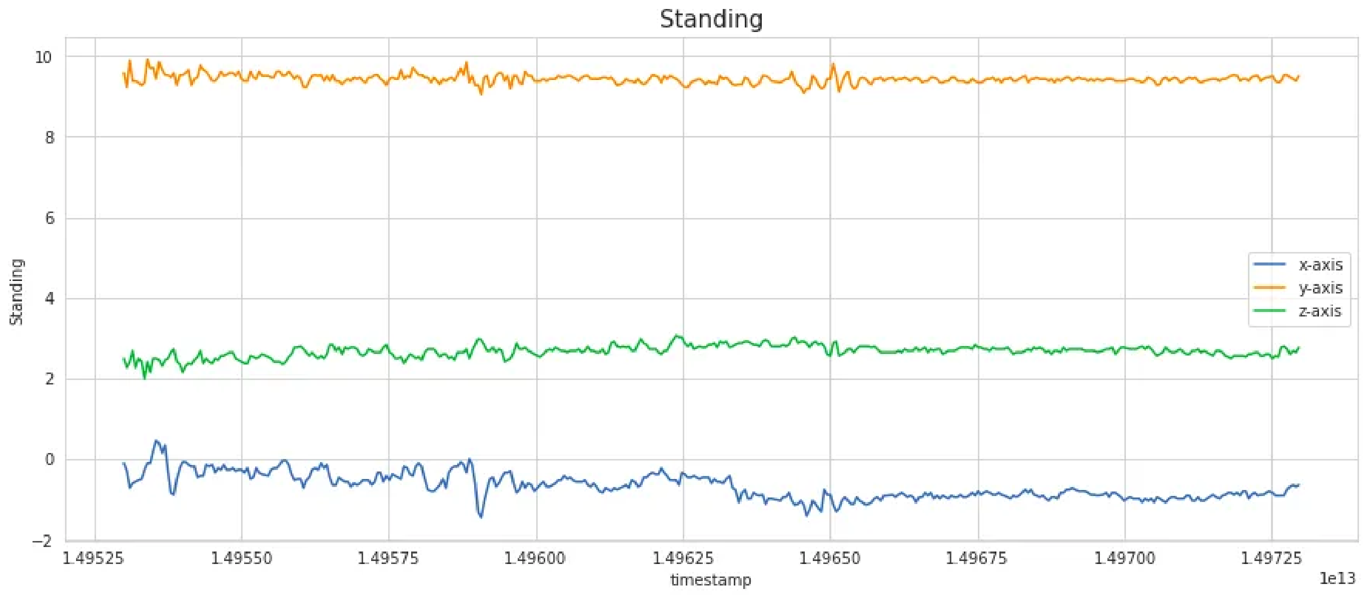
Results

Exploratory Data Analysis (EDA)



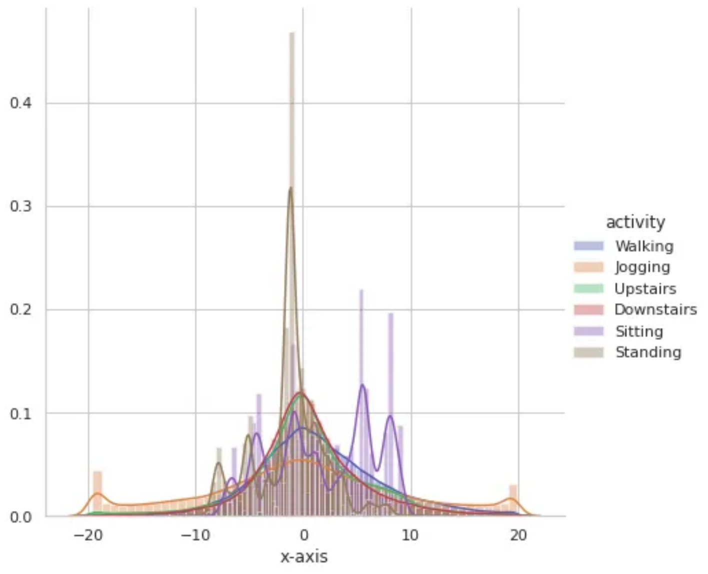


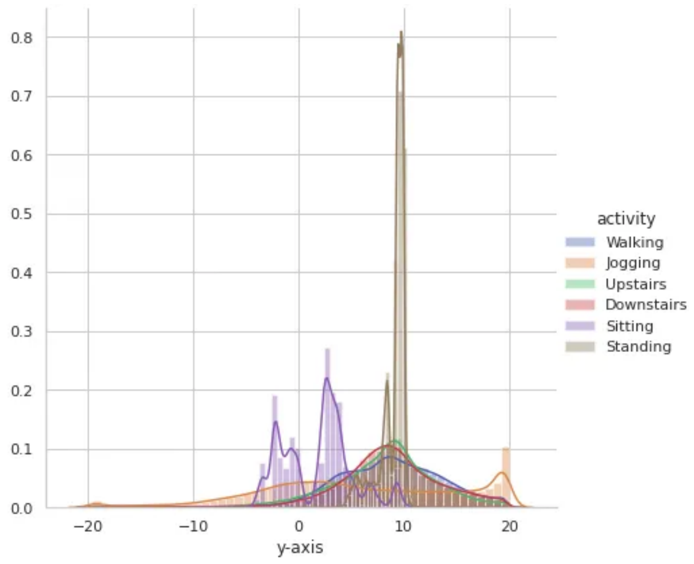


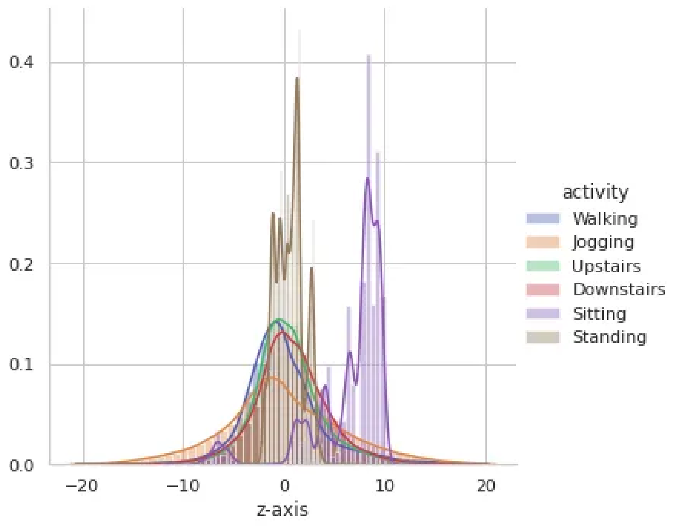


Figures 15-18

Figures 15-18 show matplotlib generated graphs of the triaxial accelerometer data for the first 400 samples of the signal. Visually, accelerometer data while the user is walking/standing have higher amplitudes and frequency than data from sitting/standing.







Figures 19-21

Figures 19-21 show the distribution of signal data for 3 axes with each activity hue, to see if there was an obvious pattern in accelerometer data between different activities. It is observed that there is very high overlap in the data among activities like Upstairs, Downstairs, Walking, Jogging and Standing on all the axes. Sitting appears to have distinctive values along the y-axis and z-axis.

Model Training with Fourier Transform

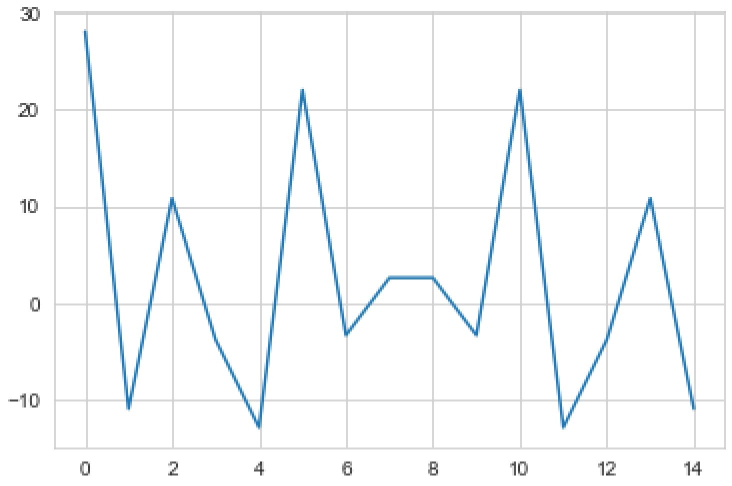


Figure 22

Figure 22 shows a fourier transform applied to the time series dataset. The DC component is the first value and is usually high, but the frequency signal is symmetrical about the center (around the 7th accelerometer value). As mentioned before, 18 key features were extracted from the second half of the fourier transform (mean, median, mode, etc) since the DC component is high and should be discarded.

Model Testing - FOG Model

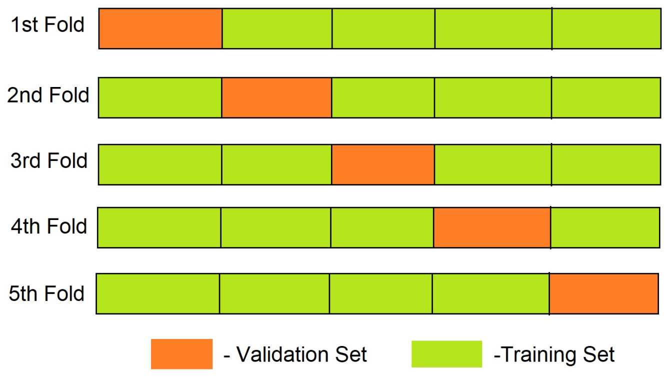


Figure 23

The FOG model was tested with a 5 fold Cross validation performed in Python. Data is typically split using the train test split (70% of the data used to train models, and the other 30% used to test the model accuracy) in classical problems, but splitting the data with a train-test split has more bias and can lead to overfitting, so k-fold splits the data in a different manner (Figure 23). K-fold cross validation involves splitting data into k equal folds (Figure 23). The first k-1 folds are used for training, and the remaining are used for testing. This is repeated for all k-folds, and the mean of the accuracies of each k-fold is returned. The 5-fold CV calculates sensitivity, specificity, F1 score, MCC, and accuracy. Sensitivity measures the ability of the model to predict FOG windows. Specificity measures the ability of the model to predict Non-FOG windows or be able to distinguish FOG from Non-FOG. The equations are shown below.

$$Sensitivity = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives(FN)}}$$

$$Specificity = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Positives(FP)}}$$

Equation 1**:** Sensitivity and Specificity measurements

F1 score is the harmonic mean of precision and recall, which measures the balance between the two metrics. It provides a single number to summarize the overall performance of a model. MCC (Matthews correlation coefficient) measures the correlation between predicted and actual binary classifications, taking into account true positives, true negatives, false positives, and false negatives. It ranges from -1 (completely incorrect) to 1 (perfectly correct), with 0 indicating no correlation. The equations for each are shown below.

$$F1 - score = \frac{2 \text{ x } (\text{precision x recall})}{(\text{precision} + \text{recall})}$$

$$\text{MCC} = \frac{\text{TP x TN} - \text{FP x FN}}{\sqrt{(\text{TP+FP})(\text{TP+FN})(\text{TN+FP})(\text{TN+FN})}}$$

Equation 2: F1 and MCC measurements

Accuracy	0.87
F1 Score	0.82
Sensitivity	0.77
Specificity	0.91

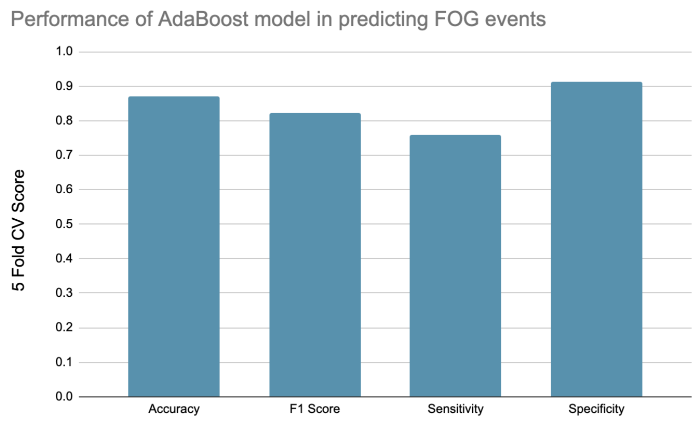


Table 1 and Figure 24: Model Performance

Table 1 and Figure 24 show the performance of the AdaBoost model in predicting FOG windows vs Non-FOG windows (windows of accelerometer data annotated during FOG events). The scores are the average of the model's performance for each fold during the 5-fold CV. The model seems to have slightly outperformed the Daphnet study in predicting FOG events, as the sensitivity of 77% is greater than the Daphnet reported sensitivity of 73%. The model also shows a fairly high accuracy of 87% in predicting FOG events. In addition to 5-fold CV, another method of cross validation was used to test the model called LOOCV (Leave-One-Out-Cross-Validation). LOOCV is a technique used for assessing the performance of an MLmodel, particularly in cases where the data sample size is relatively small. In LOOCV, the model is trained on all but one data point in the dataset, and then the performance of the model is evaluated on the one data point that was held out. This process is repeated for each data point in the dataset, and the performance of the model is averaged across all the held-out data points.This method of cross validation more accurately represents the model's performance in the real world, where it is trained on the entire dataset and only makes predictions on a single data point. However, the method is also computationally expensive (since the model has to be tested on every datapoint) so was only used to evaluate the model.

	Macro Average	Weighted Average
Accuracy	0.88	0.88
F1 Score	0.85	0.88
Sensitivity	0.85	0.88
Specificity	0.84	0.88

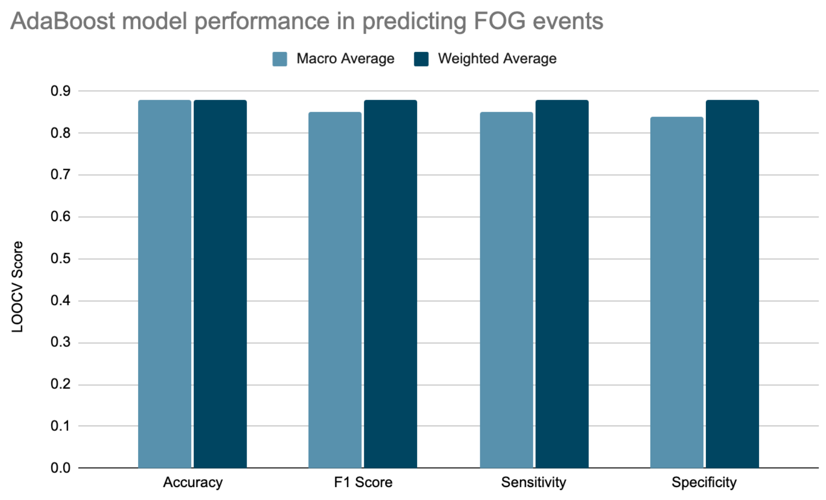
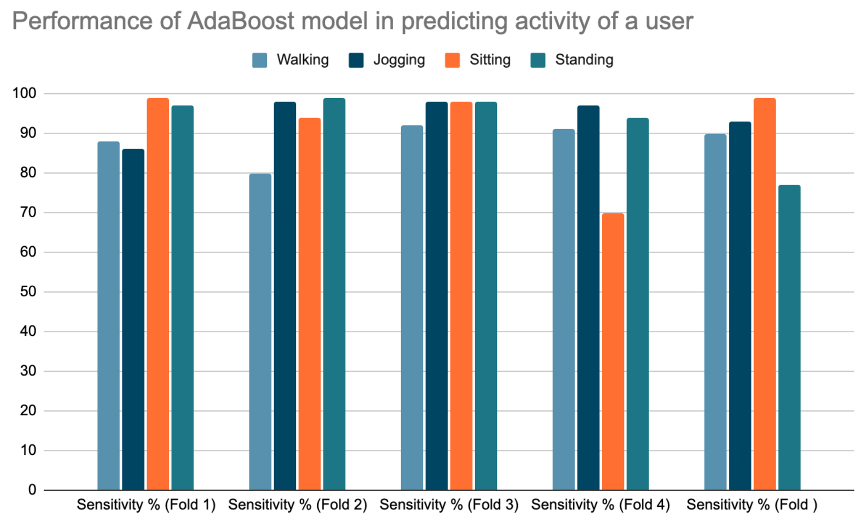
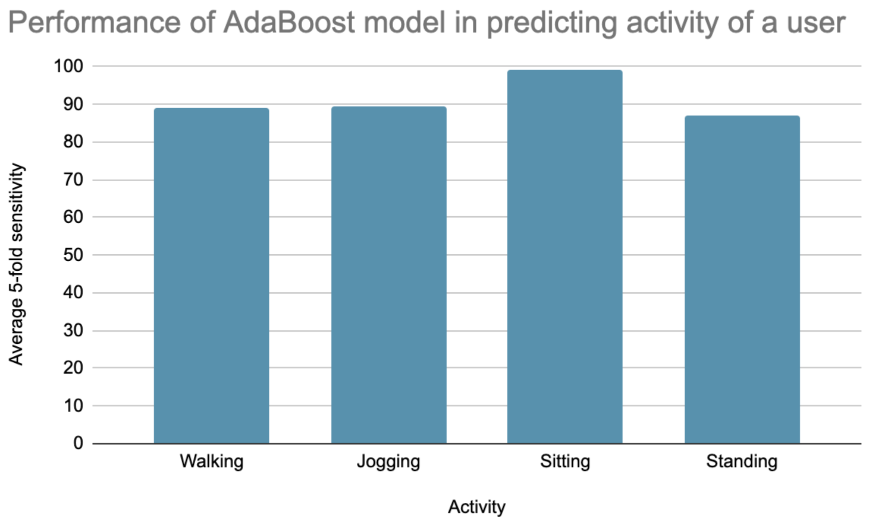


Table 2 and Figure 25: Model Performance

Upon LOOCV, the model performed well with a large improvement in sensitivity (88%) and an improvement in accuracy of 88%.

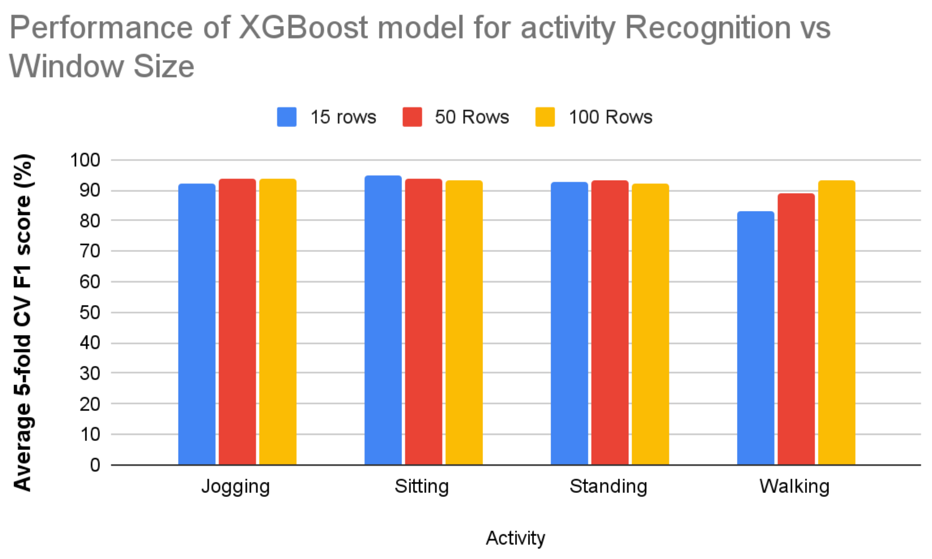
Activity Recognition Model





Figures 26 - 27

Figures 26 and 27 show the 5-fold cross validation results for the sensitivity of the XGBoost Activity Recognition model in distinguishing walking, jogging, sitting, and standing. The model showed an average 5-fold CV sensitivity of 88, 86, 99, 97 % for predicting walking, jogging, sitting, and standing respectively. This model was also tested at 3 different window lengths (15, 50, and 100) to identify the best window length. The average F1-score across 5-folds for the model in predicting each activity is shown in Figure 28.



	15 rows	50 Rows	100 Rows
Jogging	92.2	93.6	93.6
Sitting	95	94	93.2
Standing	92.6	93.4	92.2
Walking	83.4	89	93

Figure 28 and Table 3

Based on Figure 28, a higher window size clearly shows a higher F1 score when predicting walking. However, for other activities, window size seems to have a negligible effect on model performance. Table 3 shows that the model shows a high F1 score (>90%) for predicting all activities when trained on windows of 50 and 100 rows and a high F1 score in all activities except walking when trained on windows of size 15.

Conclusions and Discussions

This project successfully created a robust machine learning pipeline with an integrated and scalable iOS prototype application for activity recognition and Freezing of Gait (FOG) monitoring in Parkinson's Patients. Upon LOOCV and 5-fold cross validation, the AdaBoost machine learning model achieved an 87 and 88% accuracy respectively in classifying FOG events in the training dataset. The Activity Recognition Model with an XGBoost model framework achieved F1 scores of 93, 94, 93.4, and 89 in classifying jogging, sitting, standing, and walking respectively.

Further development of this app will benefit Parkinson's Patients by providing an accurate and reliable method of diagnosis via monitoring of Freezing of Gait, and will allow a user to keep track of daily FOG events to determine the severity of their disease symptoms. There are a few limitations with the project: 1) The use of a sliding window without overlapping windows may decrease model accuracy, since overlapping windows ensures that every subsequent row in the transformed dataset has some information from the data in the previous window as well. 2) The FOG model's accuracy can still be improved, and an active learning approach may be required for more accurate FOG detection for a specific user. 3) Only data from one subject from the original Daphnet dataset was used to train the FOG model, since the model performed poorly (< 70% accuracy) with other data or with combinations of subject data. 4) The use of smaller sliding windows (n <50) may improve app latency but may not always be able to capture the full duration of FOG events, which are usually much longer. 5) App development is still in a preliminary stage, and a UI is still yet to be created.

In the future, we plan on implementing an active learning approach where a user can manually input data on recorded FOG events to improve the model's predictive accuracy and to have the model learn the patterns of FOG in a specific user. In this study, only a baseline model to predict FOG was created and active learning may be required for more accurate detection of FOG. In addition, user testing of the app is in progress as of now, and real-life app testing data will be collected by our presentation day. Finally, the creation of an app user interface is still in progress.

Acknowledgements

We would like to thank Jeremey Jensen, our teacher, for supporting and guiding us through the project and our parents for their encouragement and support. We would also like to thank the Supercomputing Challenge Judges for giving us feedback during the midterm interview.

Bibliography

1. *Dataset*. WISDM Lab: Dataset. (n.d.). Retrieved April 4, 2023, from https://www.cis.fordham.edu/wisdm/dataset.php
2. Ensemble methods. Corporate Finance Institute. (2022, December 14). Retrieved April 4, 2023, from https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/
3. Fourier transform (ft). Questions and Answers in MRI. (n.d.). Retrieved April 4, 2023, from https://mriquestions.com/fourier-transform-ft.html
4. Kondo, Y., Mizuno, K., Bando, K., Suzuki, I., Nakamura, T., Hashide, S., Kadone, H., & Suzuki, K. (2022, April 29). Measurement accuracy of freezing of gait scoring based on videos. Frontiers. Retrieved April 4, 2023, from https://www.frontiersin.org/articles/10.3389/fnhum.2022.828355/full
5. Lutins, E. (2017, August 2). *Ensemble Methods in Machine Learning: What are they and why use them?* Medium. Retrieved April 4, 2023, from https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-an d-why-use-them-68ec3f9fef5f
6. Mayo Foundation for Medical Education and Research. (2023, February 17). Parkinson's disease. Mayo Clinic. Retrieved April 4, 2023, from https://www.mayoclinic.org/diseases-conditions/parkinsons-disease/symptoms-causes/syc -20376055
7. Nabriya, P. (2021, July 12). *Feature engineering on time-series data*. Medium. Retrieved April 4, 2023, from

https://towardsdatascience.com/feature-engineering-on-time-series-data-transforming-signal-data-of-a-smartphone-accelerometer-for-72cbe34b8a60

8. Navlani, A. (2018, November 20). AdaBoost classifier algorithms using python Sklearn tutorial. DataCamp. Retrieved April 4, 2023, from https://www.datacamp.com/tutorial/adaboost-classifier-python
9. NHS. (n.d.). NHS choices. Retrieved April 4, 2023, from https://www.nhs.uk/conditions/parkinsons-disease/treatment/#:~:text=You%20may%20not%20need%20any,and%20your%20family%20or%20carers.
10. Productionizing distributed XGBoost to train deep tree models with large data sets at uber. Uber Blog. (2019, December 10). Retrieved April 4, 2023, from https://www.uber.com/blog/productionizing-distributed-xgboost/
11. Saini, A. (2023, March 3). *Master the adaboost algorithm: Guide to implementing & understanding adaboost*. Analytics Vidhya. Retrieved April 4, 2023, from https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/
12. UCI Machine Learning Repository: Daphnet freezing of Gait Data Set. (n.d.). Retrieved April 4, 2023, from https://archive.ics.uci.edu/ml/datasets/Daphnet+Freezing+of+Gait
13. Wikimedia Foundation. (2023, January 18). *Random Forest*. Wikipedia. Retrieved April 4, 2023, from https://en.wikipedia.org/wiki/Random_forest
14. Wikimedia Foundation. (2023, March 24). Fast fourier transform. Wikipedia. Retrieved April 4, 2023, from https://en.wikipedia.org/wiki/Fast_Fourier_transform