Fitness Tracker in MatLab

***by Team* TrackStrikH**

Counting Steps

1. Preparing Data

To work with the data, we must first load it into the workspace and store the variables that are going to be used in this example. While we can double click on the data to load it, we will use the load command to make this process easier to replicate.

```
% Load the data
dat = load('Data_1.mat');

% Extract data from the struct
lat = dat.Position.latitude;
lon = dat.Position.longitude;
positionDatetime = dat.Position.Timestamp;

Xacc = dat.Acceleration.X;
Yacc = dat.Acceleration.Y;
Zacc = dat.Acceleration.Z;
accelDatetime = dat.Acceleration.Timestamp;

% We use the following to obtain linear time data in seconds from a datetime array
positionTime = timeElapsed(positionDatetime);
accelTime = timeElapsed(accelDatetime);

% Display some parts of the data
disp('Position Data Sample:');
```

```
Position Data Sample:
```

```
disp(table(lat(1:5), lon(1:5), positionDatetime(1:5)));
```

```
     Var1      Var2              Var3
    ______    _____    ________________________

    13.031    77.74    11-Jun-2024 09:57:04.000
    13.031    77.74    11-Jun-2024 09:57:05.000
    13.031    77.74    11-Jun-2024 09:57:06.000
    13.031    77.74    11-Jun-2024 09:57:07.000
    13.031    77.74    11-Jun-2024 09:57:08.000
```

```
disp('Acceleration Data Sample:');
```

```
Acceleration Data Sample:
```

```
disp(table(Xacc(1:5), Yacc(1:5), Zacc(1:5), accelDatetime(1:5)));
```

```
     Var1       Var2      Var3              Var4
    _______    ______    ______    ________________________

    -1.2258    2.9425    9.1411    11-Jun-2024 09:57:01.515
    -1.0199    2.9999    9.2129    11-Jun-2024 09:57:01.534
    -1.3048     2.897    9.1243    11-Jun-2024 09:57:01.553
    -1.0247    2.5857    9.6103    11-Jun-2024 09:57:01.572
    -0.7039    2.4253    10.238    11-Jun-2024 09:57:01.591
```

2. Determining Approach Method

Let's say that you wanted to track the number of steps taken. One way that this can be done is by calculating the total distance traveled and then dividing it by your stride length. This works because stride is the distance taken for one step so if we look at the units they cancel out as seen below.

$$\frac{\text{totalDistance(ft)}}{\text{stride}\left(\frac{\text{ft}}{\text{step}}\right)} = \frac{\text{ft}}{\frac{\text{ft}}{\text{step}}} = \text{ft} * \frac{\text{step}}{\text{ft}} = \text{steps}$$

3. Coding The Model

Function Name: `haversine`

Purpose: Calculate the great-circle distance between two points on the Earth's surface, given their latitude and longitude coordinates, using the Haversine formula.

Inputs:

- `lat1` (double): Latitude of the first point in degrees.
- `lon1` (double): Longitude of the first point in degrees.
- `lat2` (double): Latitude of the second point in degrees.
- `lon2` (double): Longitude of the second point in degrees.

Output:

- `distance_miles` (double): The distance between the two points in miles.

```
function distance_miles = haversine(lat1, lon1, lat2, lon2)
    % Convert degrees to radians
    lat1 = deg2rad(lat1);
    lon1 = deg2rad(lon1);
    lat2 = deg2rad(lat2);
    lon2 = deg2rad(lon2);

    % Haversine formula
    dlat = lat2 - lat1;
    dlon = lon2 - lon1;
    a = sin(dlat/2)^2 + cos(lat1) * cos(lat2) * sin(dlon/2)^2;
    c = 2 * atan2(sqrt(a), sqrt(1-a));

    % Earth's radius in miles
    earthRadius_miles = 3958.8; % average radius of Earth in miles

    % Calculate distance in miles
    distance_miles = c * earthRadius_miles;
end
```

Formula

$$\frac{\text{traveledDistance}}{\text{earth'sCircumference}} = \frac{\text{degreesTraveled}}{\text{earth'sDegrees}}$$

From this we solve for 'traveledDistance'. We will use 360 for 'earth’sDegrees' because that is the total number of degrees in a circle. While the earth is not a perfect circle, this will still be a good estimation

Steps :

Define Constants:

- Set the Earth's circumference in miles.
- Set the average stride length in feet.

Initialize Total Distance:

- Initialize a variable to store the accumulated total distance traveled.

Loop Through Each Pair of Points:

- Iterate through the latitude and longitude data points.
- For each pair of consecutive points:
- Extract the current and next latitude and longitude.
- Calculate the distance between the two points using the Haversine formula.

- Accumulate this distance into the total distance variable.

Convert Total Distance to Feet:

- Convert the total distance from miles to feet.

Calculate the Total Number of Steps:

- Calculate the number of steps by dividing the total distance in feet by the average stride length.

Display the Results:

- Print the total distance traveled in miles.
- Print the total number of steps taken.

```
%disp(['The total distance traveled is: ', num2str(totaldis),' miles'])
%disp(['You took ', num2str(steps) ' steps'])
earthCirc = 24901; % Earth's circumference in miles
stride = 2.5; % Average stride length in feet

% Initialize total distance
totaldis = 0;

% Loop through each pair of points
for i = 1:(length(lat)-1)
    lat1 = lat(i);     % The first latitude
    lat2 = lat(i+1);   % The second latitude
    lon1 = lon(i);     % The first longitude
    lon2 = lon(i+1);   % The second longitude
    degDis = haversine(lat1, lon1, lat2, lon2);
    totaldis = totaldis + degDis;
end

% Convert total distance to feet
totaldis_ft = totaldis * 5280;

% Calculate the total number of steps
steps = totaldis_ft / stride;

% Display the results
disp(['The Total Distance Traveled is: ', num2str(totaldis), ' Miles'])
```

```
The Total Distance Traveled is: 0.1953 Miles
```

```
disp(['You Took ', num2str(steps), ' Steps'])
```

```
You Took 412.4687 Steps
```

Classifying Activity

In addition to counting steps, let's say that you want to know what kind of activity you were doing. Let's train a machine learning model that takes a person's acceleration data and shows if they were sitting, walking, or running.

1. Prepare the data

First, We need to collect sensor data for each of these activities. For simplicity, I collected this data in three different sessions: one where I was sitting the whole time ('sitFile'), one where I was walking ('walkFile'), and one where I was running ('runFile').

```matlab
% File paths for the activity log .mat files
runFile = "runningPositionAcceleration.mat";
sitFile = "sittingPositionAcceleration.mat";
walkFile = "walkingPositionAcceleration.mat";

% Load running data
runData = load(runFile);
runLogs = runData.Acceleration;
runLogs.Activity = repmat("running", size(runLogs, 1), 1); % Add activity label

% Load sitting data
sitData = load(sitFile);
sitLogs = sitData.Acceleration;
sitLogs.Activity = repmat("sitting", size(sitLogs, 1), 1); % Add activity label

% Load walking data
walkData = load(walkFile);
walkLogs = walkData.Acceleration;
walkLogs.Activity = repmat("walking", size(walkLogs, 1), 1); % Add activity label

% Combine all logs into a single structure or table
allLogs = [runLogs; sitLogs; walkLogs];

% Save combined data to a single .mat file
save("combinedActivityLogs.mat", "allLogs");

% Alternatively, you can continue to work with allLogs for further analysis
```

Let's load the logs for these sessions and preview one of them to get a sense of how the data is structured.

```
% Load data into MATLAB workspace (assuming allLogs contains the data)
load("combinedActivityLogs.mat", "allLogs");
disp(allLogs)
```

```
          Timestamp                  X           Y           Z         Activity
    _______________________    ________    ________    ________    ________

    19-Jun-2024 14:06:56.948    -0.56052     2.6734      9.6855     "running"
    19-Jun-2024 14:06:57.049    -0.51864     2.5663      9.7603     "running"
    19-Jun-2024 14:06:57.150    -0.46421     2.7655      9.6455     "running"
    19-Jun-2024 14:06:57.251    -0.46301     2.8857      9.3817     "running"
    19-Jun-2024 14:06:57.352    -0.57727     2.8337      9.6227     "running"
    19-Jun-2024 14:06:57.453    -0.37328     2.7709       9.655     "running"
    19-Jun-2024 14:06:57.553    -0.35414     2.7506      9.5946     "running"
    19-Jun-2024 14:06:57.655    -0.42951     2.7805      9.4642     "running"
    19-Jun-2024 14:06:57.756    -0.45224     3.0443        9.28     "running"
    19-Jun-2024 14:06:57.857    -0.30628     3.0772      9.5791     "running"
    19-Jun-2024 14:06:57.958    -0.38285     2.9109      9.3601     "running"
    19-Jun-2024 14:06:58.059    -0.27817     2.9587      9.5055     "running"
```

```
% Assuming allLogs is structured with fields like 'Acceleration', 'Timestamp', etc.
% Extract data for sitting, walking, and running
sitAcceleration = allLogs(allLogs.Activity == "sitting", :);
walkAcceleration = allLogs(allLogs.Activity == "walking", :);
runAcceleration = allLogs(allLogs.Activity == "running", :);

% Define activity labels
sitLabel = 'sitting';
walkLabel = 'walking';
runLabel = 'running';

% Convert labels to cell array of character vectors
sitLabel = cellstr(repmat(sitLabel, size(sitAcceleration, 1), 1));
walkLabel = cellstr(repmat(walkLabel, size(walkAcceleration, 1), 1));
runLabel = cellstr(repmat(runLabel, size(runAcceleration, 1), 1));

% Assign categorical labels to each subset of data
sitAcceleration.Activity = categ    orical(sitLabel);
walkAcceleration.Activity = categorical(walkLabel);
runAcceleration.Activity = categorical(runLabel);

% Combine all data into one table
allAcceleration = [sitAcceleration; walkAcceleration; runAcceleration];
% Remove 'Timestamp' variable (if it exists)
if ismember('Timestamp', allAcceleration.Properties.VariableNames)
    allAcceleration.Timestamp = [];
end

% Convert timetable to table (if needed)
allAcceleration = timetable2table(allAcceleration, "ConvertRowTimes", false);
% Now the data is ready to be used for training
disp("Data preparation complete.");
```

```
Data preparation complete.
```

```
disp("Size of allAcceleration table:");
```

```
Size of allAcceleration table:
```

```
disp(size(allAcceleration));
```

```
        7037           4
```

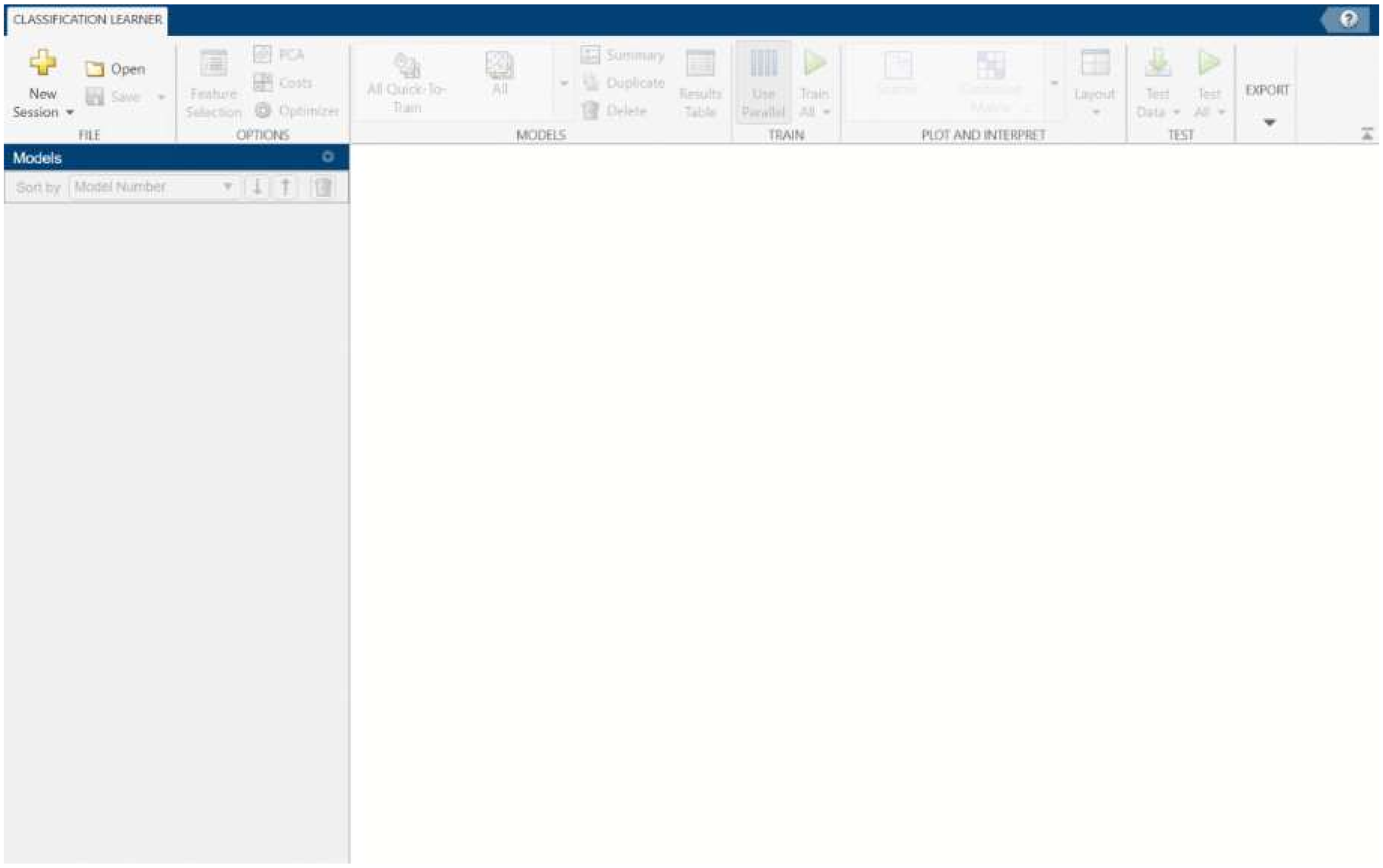
2. Train a model

One way to train several different machine learning algorithms at once is to use the Classification Learner App, which allows you to rapidly prototype and fine tune your models until you have one you are happy with. You have a lot of options on how to use your data and train models in this app, so while this will be a basic workflow, you can learn more about the different options using this documentation.

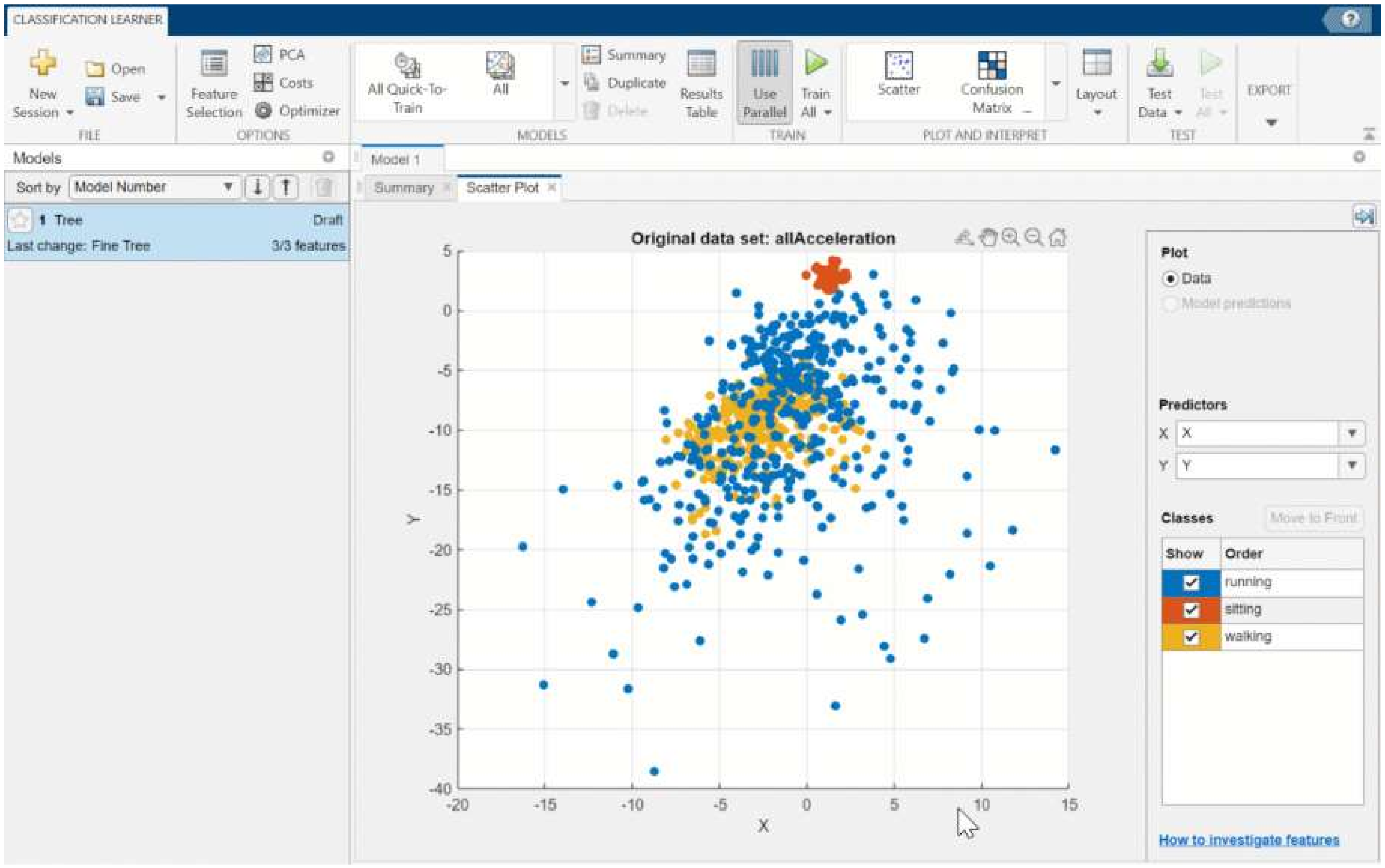
First, open the app:

```
classificationLearner;
```

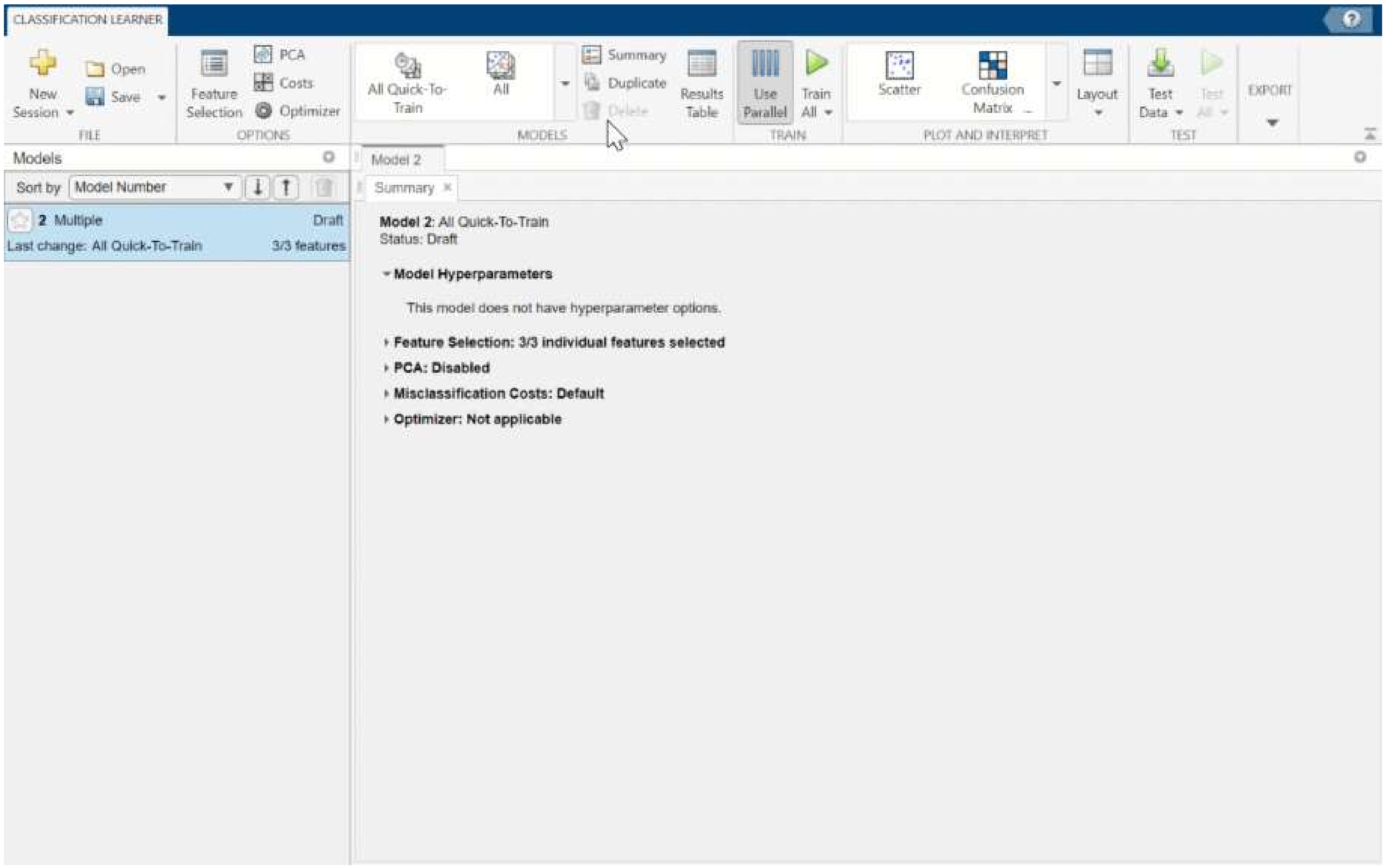
Start a "New Session" and select "From Workspace". In the window that opens, select `allAcceleration` as the 'Data Set Variable', then start the session.



From the 'Models' section of the toolstrip, select 'All Quick-To-Train'. This will add a model labeled 'Multiple' to the 'Models' tab. Then delete the model labeled 'Tree'.



Click 'Train All', which will start training several models. Once the training has finished, each model will have an 'RMSE (Validation)' value. This represents the *root mean squared error* of that model when used on a subset of the training data. The lower the RMSE, the better the model performed. Select the model with the lowest RMSE, then export the model to the workplace.



Model 2: All Quick-To-Train
Status: Draft

Model Hyperparameters

This model does not have hyperparameter options.

Feature Selection: 3/3 individual features selected

PCA: Disabled

Misclassification Costs: Default

Optimizer: Not applicable

3. Use the model

Now we can use that model to make predictions on new data! 'Data.mata' is a log of Acceleration, Position data collected during a sample workout that included walking, running, and sitting, so we can use our machine learning model to predict what activity the person was doing for each timestamp.

```
% Load the test data
data = load("Data.mat");

% Display the contents of the loaded data
%disp(data);

% Extract the acceleration data from the test data
testAcceleration = data.Acceleration;
disp(testAcceleration)
```

```
           Timestamp               X          Y         Z
    ________________________    _______    ______    ______

    19-Jun-2024 12:26:19.667    0.77895     1.341    10.883
    19-Jun-2024 12:26:19.717      0.501     1.247    9.5831
    19-Jun-2024 12:26:19.767    0.92505     1.219     10.11
    19-Jun-2024 12:26:19.817      0.561    1.0871     9.839
    19-Jun-2024 12:26:19.867    0.75495     1.113    9.7271
    19-Jun-2024 12:26:19.917    0.82905     1.185       9.6
    19-Jun-2024 12:26:19.967      0.849     1.226     9.511
    19-Jun-2024 12:26:20.017      0.612     1.049    10.411
    19-Jun-2024 12:26:20.067    0.79395     1.367    9.4991
    19-Jun-2024 12:26:20.117    0.88695    1.4269    10.105
    19-Jun-2024 12:26:20.167    0.79605     1.429     9.861
    19-Jun-2024 12:26:20.217       1.16     1.607    9.5741
```

```

% Convert the timetable to a table and remove the 'Timestamp' variable (if it exists)
testTable = timetable2table(testAcceleration, "ConvertRowTimes", false);

% Ensure the table has the same structure as the training data (excluding the response variable)
% For example, if the training data had columns 'X', 'Y', 'Z' for acceleration
predictorVars = {'X', 'Y', 'Z'};

% Select the necessary predictors from the test table
testPredictors = testTable(:, predictorVars);

% Display the prepared test predictors to verify
disp("Prepared test predictors:");
```

```
Prepared test predictors:
```

```
disp(testPredictors);
```

```
       X          Y         Z
    _______    ______    ______

    0.77895     1.341    10.883
      0.501     1.247    9.5831
    0.92505     1.219     10.11
      0.561    1.0871     9.839
    0.75495     1.113    9.7271
    0.82905     1.185       9.6
      0.849     1.226     9.511
      0.612     1.049    10.411
    0.79395     1.367    9.4991
    0.88695    1.4269    10.105
    0.79605     1.429     9.861
       1.16     1.607    9.5741
```

```

% Load the trained model (assuming trainedModel is already in the workspace or load it from a file)
% load('trainedModel.mat'); % Uncomment if you need to load the model from a file

% Use the trained model to make predictions on the test data
predictions = trainedModel.predictFcn(testPredictors);

% Display the predictions
disp("Predictions for the test data:");
```

```
Predictions for the test data:
```

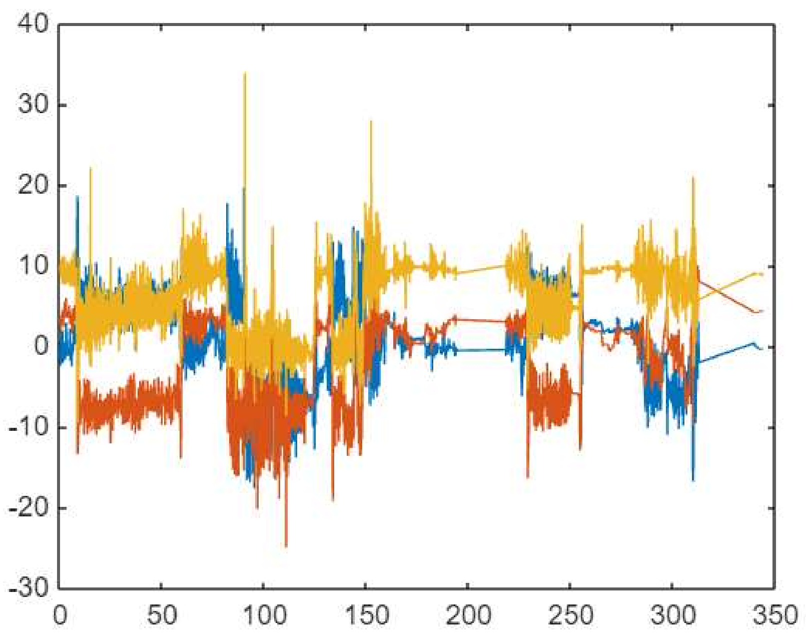
```
disp(predictions);
```

```
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
    walking
```

Creating Graphics

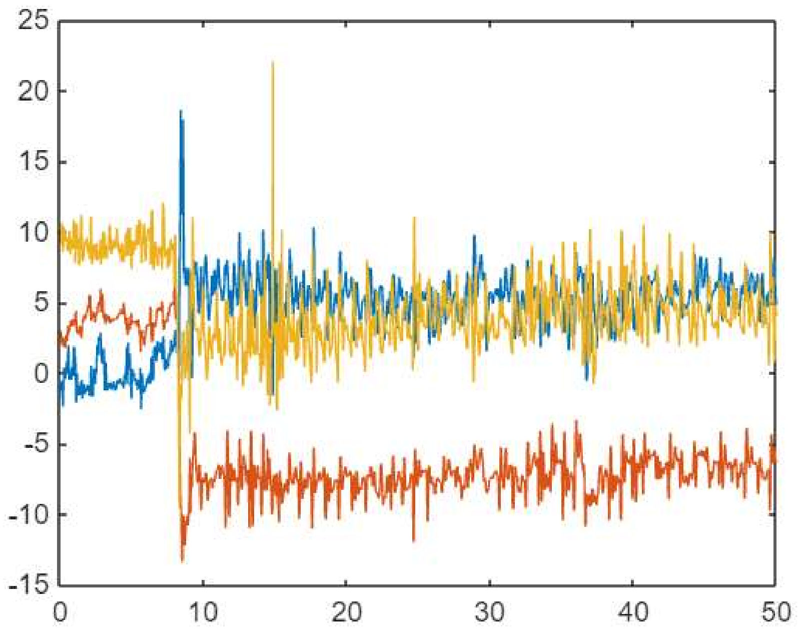
Let's visualize data in MATLAB. Say we wanted to graph the acceleration data that we took earlier to compare how the X, Y and Z data compare against one another. We can do this using the `plot` command. The only problem with this command is plotting multiple data on the same graph the code can be messy in one line. Because of this, we use the `hold on` command to plot multiple lines on the same graph. Don't forgot to also use `hold off` when you are done or else everything you plot will go on the same graph.

```
plot(accelTime,Xacc);
hold on;
plot(accelTime,Yacc);
plot(accelTime,Zacc);
hold off
```



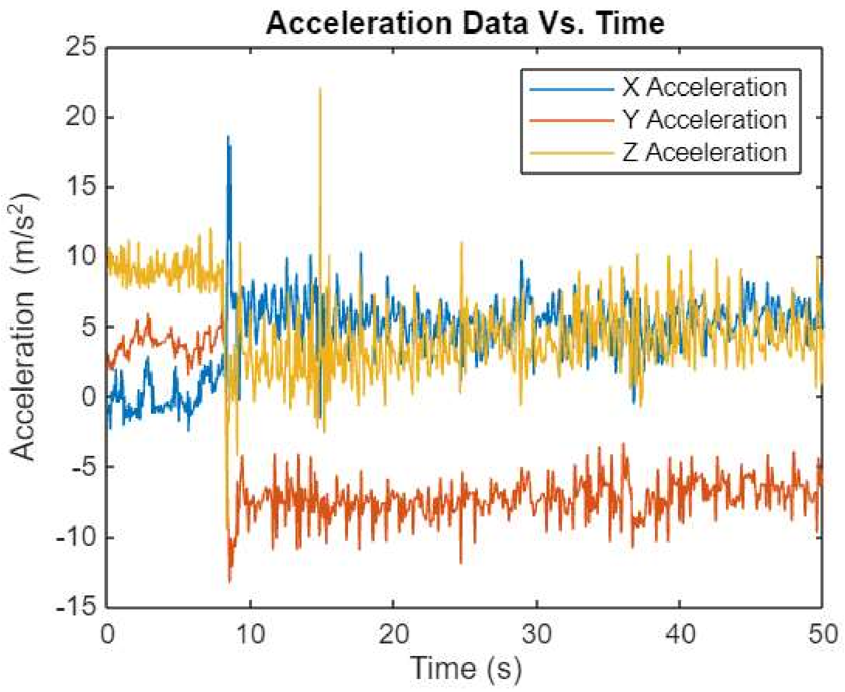
In the graph we just made it is very difficult to differentiate between the lines because there is so much data being graphed in such a small window. We will want to zoom in on the graph in order to see any trends between the different data lines. To do this, we set a limit on the x axis usng `xlim` so that we can look at the first 50 seconds of data.

```
plot(accelTime,Xacc);
hold on;
plot(accelTime,Yacc);
plot(accelTime,Zacc);
xlim([0 50])
hold off
```



Adding Title and Labels to the Graph (Final Plot)

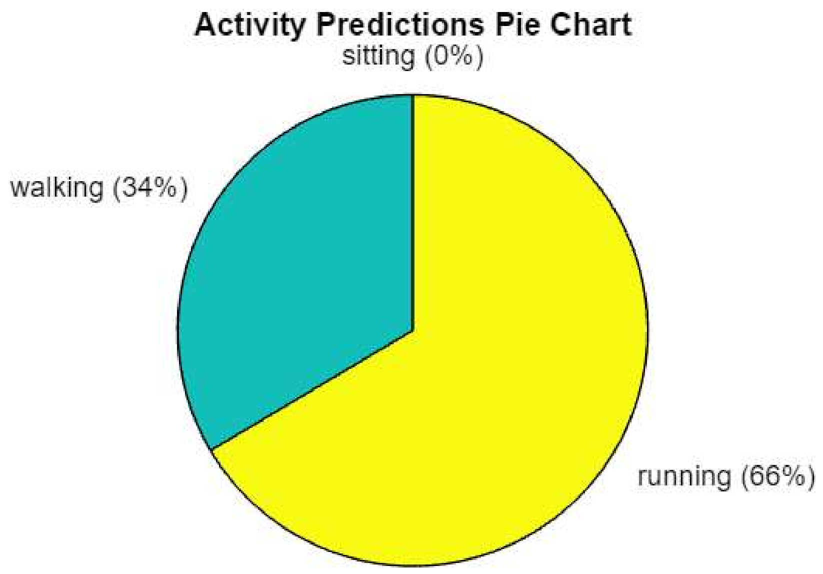
```
plot(accelTime,Xacc);
hold on;
plot(accelTime,Yacc);
plot(accelTime,Zacc);
xlim([0 50])
legend('X Acceleration','Y Acceleration','Z Aceeleration');
xlabel('Time (s)')
ylabel('Acceleration (m/s^2)');
title('Acceleration Data Vs. Time');
hold off
```



Creating a "Pie Chart" for displaying the Summary of the Activities completed during a Workout :

```
% Convert predictions to categorical array for plotting
yfitcat = categorical(predictions);

% Plot the predictions using a pie chart
figure;
pie(yfitcat);
title('Activity Predictions Pie Chart');
```



THE END