Machine Learning Approaches to Leg Motion Classification and Feature Optimization Analysis

1 INTRODUCTION

Modeling human movements, including basic leg activities such as walking, running, and jumping provides foundation for designing fitness apps, healthcare products, and locomotive robots. Leg activity recognition through machine learning classifiers enables users to analyze and describe their daily activities as models with which the computer program can then independently adapt to according to the user. Machine learning is reliable because it can provide solutions to complex problems that cannot be conventionally solved, including the documentation of rare genetic disorders that impact leg motion. In our study, we applied two machine learning classifiers, Deep Convolutional Neural Network and the Gauss Naive Bayesian classifier, followed by an in-depth analysis of feature optimization in order to train our dataset.

1.1 Motivation for Classifying Leg Movements.

The primary motivation for conducting leg movement research with machine learning was to obtain a baseline analysis of leg motions between individuals for health documentation purposes. Almost all humans walk, run, or jump in similar manners, so these similarities could be modeled by our machine learning classification method. With this, we may also identify characteristic recognitions in common leg activities that can be used to seek out leg-related issues in the case of oddities. Moreover, leg activity can also be modeled and then applied to fields that are unrelated to the medical side, such as in motion-capture in the film and animation industry to depict fluid anthropomorphic movements or in athletic science, where improving differential motions of footwork may serve to improve an athlete's performance.

1.2 Challenges in Statistics and Machine Learning Approaches.

Like with many models based on real-life movements, the machine learning approach to leg motion analysis can lead to complications from variable traits by the population or the environment. Machine learning is often used as an efficacious method to analyze data and build reliable models but it can still result in biases. For example, feature extraction from a limited data set and limited time interval per leg motion introduces biases in training, which may differ with updates to the sample size. We did not account for variables like muscle fatigue nor an individual's pre-existing health condition in influencing performance. One reason that these challenges may arise from feature extraction is because features are mathematical characteristics specific to that dataset. When we use sensors like an accelerometer to record leg motion, there are potential issues with using features for activities that share similar characteristics or have a lot of outliers. In our case, using statistical features such as standard deviation and mean, which we later analyzed, resulted in inaccurate training sets due to influence from outliers or the occurrence of noise. To mitigate these challenges, we used the classifiers to group our data into regular training datasets as well as validation datasets, then we applied feature enhancements like smoothing, which would make the enhanced data less prone to biases and fluctuations provided a limited sample.

2 DATA COLLECTION

For our data collection method, we focused on using a sensor-based accelerometer to measure lower body acceleration through walking, running, and jumping from the Phyphox mobile application, without accounting gravitational acceleration effects in the measurements. The recorded directional time series for the acceleration data include: x-direction (horizontal phone screen axis), y-direction (vertical phone screen axis), z-direction (axis from the front to the back of the screen), and absolute acceleration. In the experiment, the mobile device was mounted on the thigh region with tight pockets, straps, or adhesives in an attempt to record the most amount of leg activity per person as possible. The procedure for trials of walking, running, and jumping went as followed: we set a timed run of 60 seconds and performed the activity around a flat terrain (either indoors or outdoors) until the end of the timer was reached, then exported the results as a raw excel file to be used in the following analysis sections.

3 MACHINE LEARNING METHODS

For our machine learning methods, we decided to implement two different machine learning algorithms to train given the same input data we collected. Since the data we collected had a very small sample size, our accuracies only reflect the accuracy in classifying leg movements for the sample of people we collected data from. The actual test error when classifying leg movements for an arbitrary sample of people would likely be much less given our trained models. In order to gauge the maximum accuracy we could obtain for a machine learning model, we first chose to analyze a Deep Convolutional Neural Network. Afterwards, we applied the Gauss-Naive Bayesian classifier and compared it with the CNN model to gauge scenarios for when one is more effective than another.

3.1 Deep Convolutional Neural Network

The deep CNN method is relatively complex but yields highly accurate results for properly tuned models, thus making it a good measure for the achievable accuracy. A CNN is commonly used to detect shared features amongst images in order to correctly classify them. In our case, we use a CNN to identify common features in signals (our acceleration data over time) in order to accurately predict classes.

3.1.1 Implementation and Results. For our Deep Convolutional Neural Network, we designed the following network architecture:

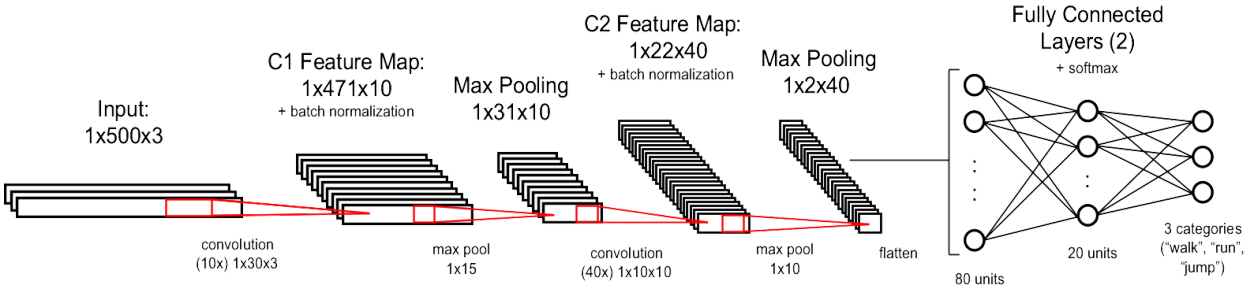


Figure 1: CNN Network Architecture Map for Layers of Leg Motion Categories

Here, we defined the input as 500 0.01(s) samples of acceleration data, where there are 3 channels for x, y, and z. The network was designed to have two layers of convolutions with max pooling, and two fully connected layers. The two layers of convolutions identify features, with max pooling, reducing, overfitting, and dimensionality. The fully connected layers then identify which features would best serve as classifications to make an accurate model. To develop this model, we had to adjust the hyperparameters listed across the bottom of the network architecture to find the most accurate model. Using this model, the total number of parameters we had to train was 6833, which is relatively small for a deep network thus making it have a relatively low time and space complexity. When training the model on the raw data with a 20% holdout we were able to achieve an average test accuracy of 90.06%, a precedingly high accuracy. Then to further improve the accuracy of the model, we trained the model with feature enhancements on the raw data, using convolutions with Gaussian, Sobel, and Difference of Gaussian filters. Using this method, we were able to achieve an accuracy of 91.46%.

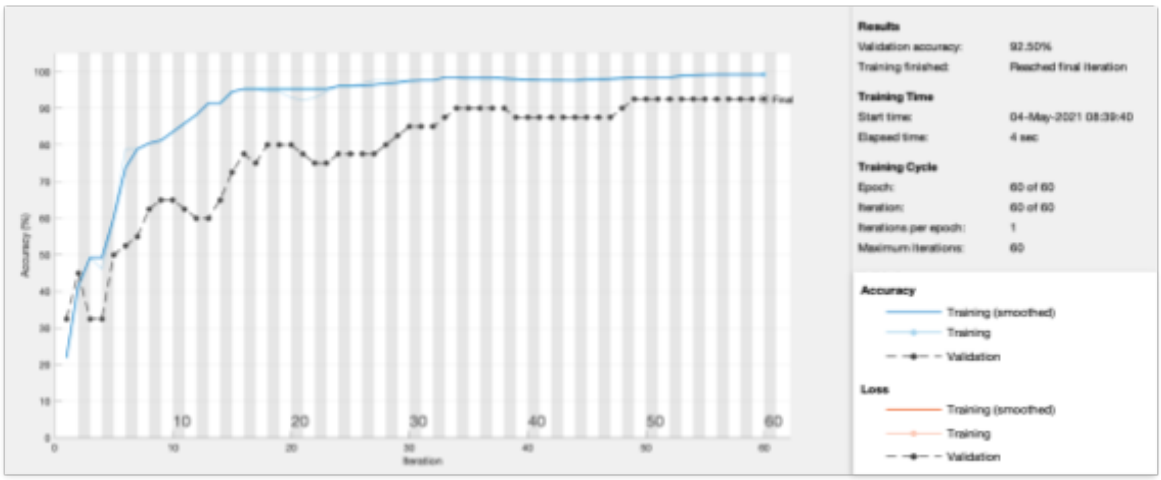


Figure 2: Plot of Sample Accuracy Curve when Training with Raw Data

3.1.2 Weighing Advantages and Drawbacks. Some advantages of using a deep CNN include shift invariance and high accuracy. Since the data does not sample specific movements at set times, our goal is to recognize common features in the data without respect to time. Thus, shift variance allows us to classify the data without any preprocessing to adjust sampling times. In addition, the deep CNN is highly accurate due to its hyperparameters being fine tuned to maximize test accuracy. Some drawbacks of the deep CNN model are that it takes time to adjust the hyperparameters, and it likely takes a long time to train, especially for larger datasets. In addition, this model requires the input to be a fixed size and needs the entire dataset in order to train the model. Thus, the classifier cannot take in additional data and update the model in real time, it has to retrain the entire model accordingly.

3.1.3 Discussion. The accuracy results of our Deep Convolutional Neural Network provides a fairly high assessment of how well we can classify our given set of data. This shows that a successful machine learning model for this dataset should be able to achieve a test accuracy of around 90%. This model worked well for our small dataset, because we were classifying already collected data. However, in many typical applications of a classifier, we want to classify data as we collect it. That is why we also did an in depth analysis of a Gaussian Naive Bayes Classifier.

3.2 Gauss Naive Bayes Classification

The Gauss Naive Bayes classifier is a simple probabilistic classification method that relies on the Bayes theorem and assumes the data is stochastically independent and normally distributed. It is a relatively simple classification technique but has high functionality when implemented with continuous data. In this section, we explored the results of the Gauss Naive Bayes classification method with features and analyzed the accuracy results when compared to the CNN technique.

3.2.1 Pre-processing Data. In order to classify the data, we first had to pre-process the raw data we collected from Phyphox. To do this, we needed the data to have the same size. One of the issues that we encountered was high noise levels in the data due to the sensitivity of the accelerometer. To solve this, we built a moving average filter with a window of 15 data points. Then, to generate data in a format the classifier could read, we created a table for each data set with each acceleration direction as a column against time, with a final column tagging each row with the mode of movement. The acceleration direction in the table's were interpreted by the classifier as four independent features, and labeled (walking, running, jumping) for the classifier.

3.2.2. Implementation and Initial Results. Once the pre-processing stage was complete, we had to decide the validation scheme we wanted in order to train our data and measure accuracy. For this particular classifier, we chose to use a holdout validation scheme. This scheme works well with large sets of data, and also tends to produce quicker results than other validation methods. Since our raw dataset was very large, consisting of over 70,000 individual data points, this method would work well with our model. To implement it, we had our classifier utilize 80% of the data for training (training dataset) and 20% for testing (testing data set). Our GNB classifier utilizes statistics and probability functions internally to train the data using only the training dataset and then assessed using the testing set of data to produce accuracy rate and error. Fig. 1 shows a visual representation of our results utilizing our modified raw data, with the motion classes color coded. Yellow represents walking, orange represents running, and blue represents jumping. An 'o' depicts a particular data point that was correctly classified. An 'x' represents a data point that was incorrectly classified. Our classifier produced initial results of about a 77% rate of accuracy.

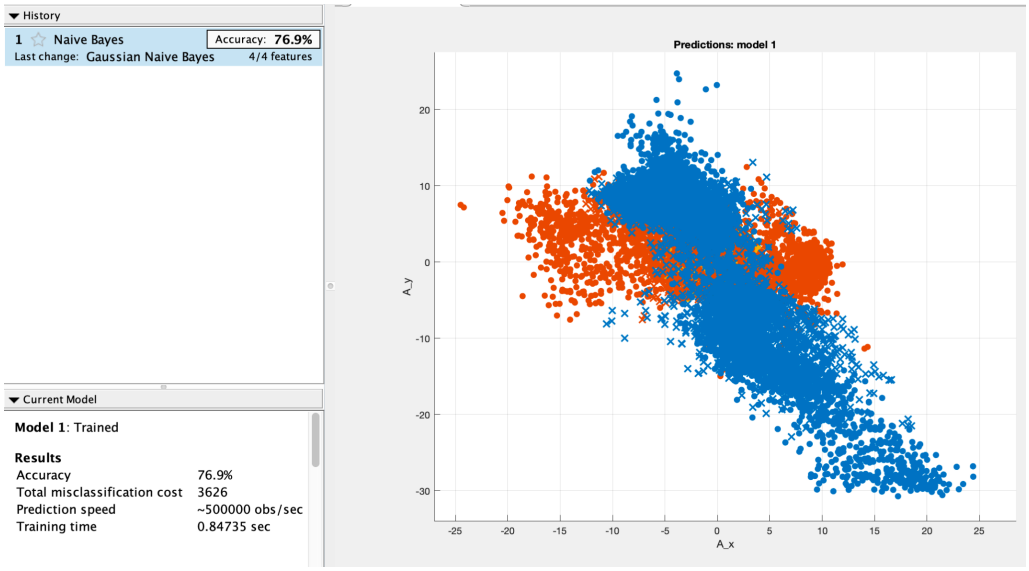


Figure 3: Classifier predictions with raw data; x axis depicts accelerometer data in the x-direction, y axis depicts accelerometer data in the y-direction

3.2.3 Optimization Using Data Features. In order to further optimize our classifier and generate better rates of accuracy, we chose to utilize features. Features are mathematical characteristics observed in a particular data set. These characteristics are used instead of the original raw data set, with the goal of finding the feature that best relays a trend within the data. We used prior knowledge of statistical parameters to choose our first two features: mean and standard deviation. To implement these, we used the MATLAB functions mean and std, which both take in an array of data and output the mean and standard deviation, respectively. We applied these functions to each individual array of data (excluding time) and compiled these results into a table again, organized by accelerometer motion and class. It is notable that the data size for these two features have greatly reduced to under 20 points of data. As we discussed above, a large data set is crucial for a holdout validation scheme to work properly. Utilizing a small data set caused a great amount of variance in the accuracy rate produced and a high amount of false positives. In order to prevent this, we used a k-fold cross validation scheme instead of holdout. The k-fold cross validation scheme divides the data into a certain amount of subsets (k) and repeats the holdout method k times. In order to keep our methods similar and to be able to compare our results to our initial classifier, we used a five fold cross validation scheme ($k = 5$) to hold out twenty percent of the data for testing as we did previously. This produced a result of about a 44% rate of accuracy for both statistical features. The accuracy rate using statistical features significantly decreased compared to our initial results because the data for these features didn't produce enough of a varying result which thus makes our classifier have a harder time spotting a trend. Hence, using mean and standard deviation would not be effective for our model.

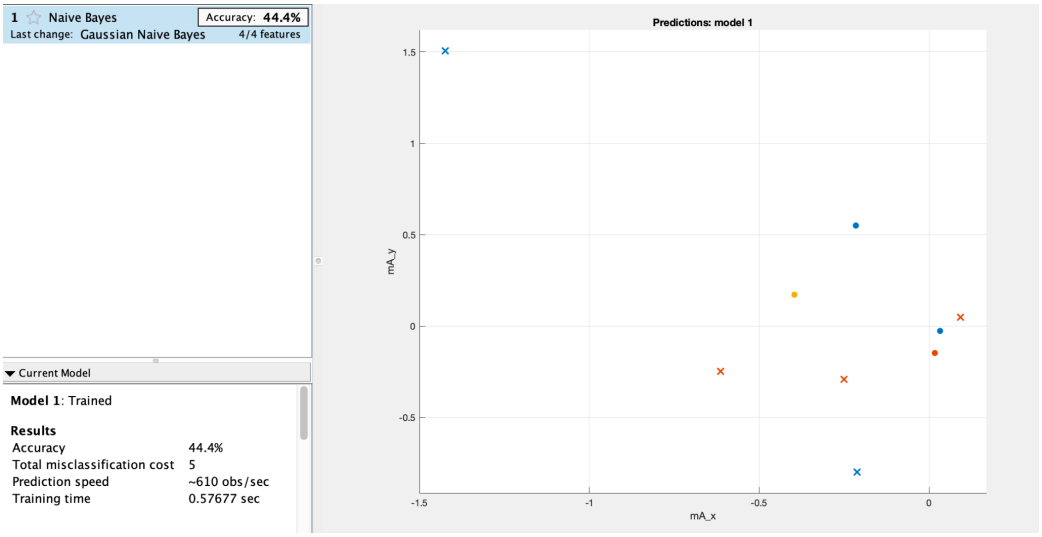


Figure 4: Classifier predictions with *mean* ($\bar{x}$) data; x-axis depicts $\bar{x}$ accelerometer data in the x-direction, y-axis depicts $\bar{x}$ accelerometer data in the y-direction

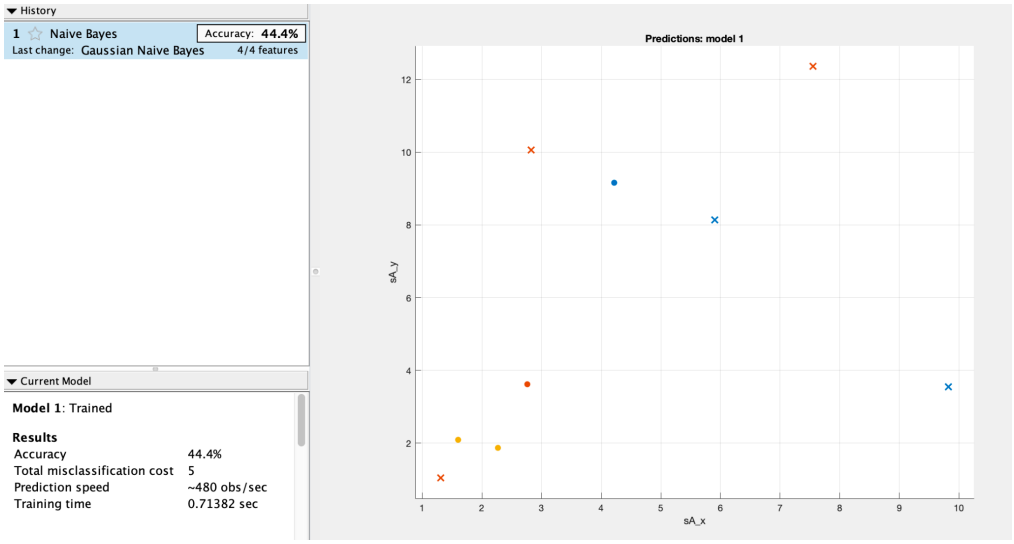


Figure 5: Classifier predictions with *standard deviation (σ)* data; x axis depicts σaccelerometer data in the x-direction, y axis depicts σ accelerometer data in the y-direction

To choose a final feature more suited towards our model, we studied the plot of each motion type and saw a visible difference in peak location. Accordingly, we chose to use peak location as a final feature to try and optimize our model. In order to implement this feature, we wrote a function to take in both accelerometer and time data as arrays and output peak location in two dimensions. One of the issues we encountered was that the amount of peaks differed for each independent motion direction. In order to fix this and make the data impartially classifiable, we revised our peak function to take in an extra variable: minimum amplitude. Using this, we had the function output the first 20 peaks above the minimum amplitude by utilizing a while loop. Once this was achieved, we ran the function for each independent data set and then compiled the data into one table. Since the data set was large, we were able to once again use the holdout method in the same fashion as previously in order to assess the accuracy of the classifier. This produced a result of about a 86.1% rate of accuracy. The accuracy rate using this feature improved compared to our initial results, meaning this feature is suited for optimization of our model. We decided to further improve our model for peak location to see if our classifier could produce even better results. In order to achieve this we used numerical analysis techniques (discussed in section 4) to create a more efficient model for peak analysis. Running our peaks from this model into the classifier resulted in a about a 90.6% percent rate of accuracy. This was the most optimal result so far for our classifier.

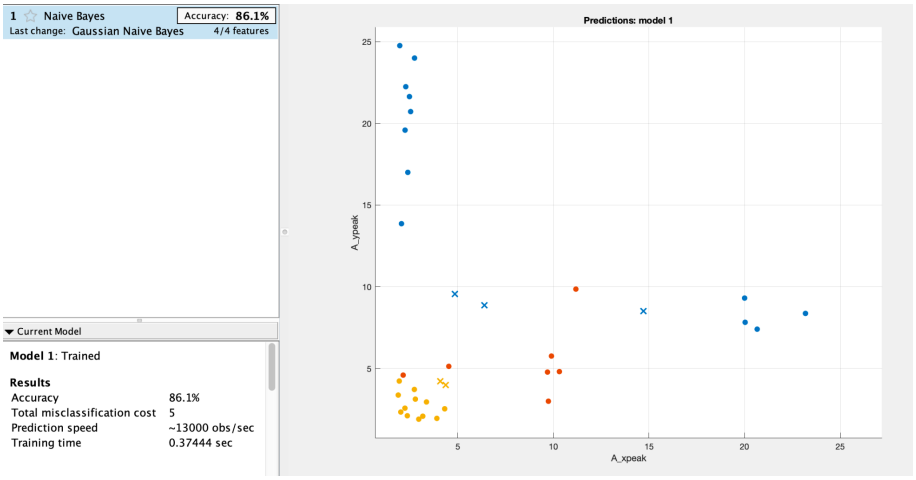


Figure 6: Classifier predictions with *peak* data on original model; x-axis depicts peak accelerometer data in the x-direction, y-axis depicts peak accelerometer data in the y-direction

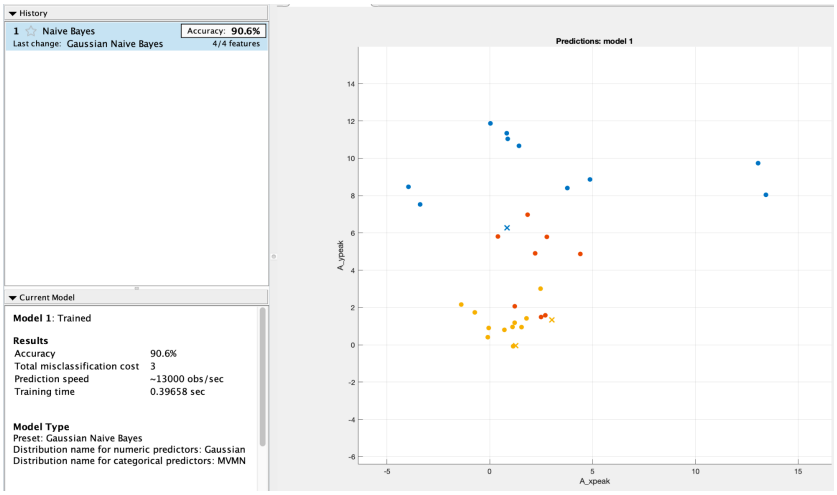


Figure 7: Classifier predictions with *peak* data on optimized model; x-axis depicts peak accelerometer data in the x-direction, y-axis depicts peak accelerometer data in the y-direction

3.2.4 Discussion. Running the classifier on feature data in this section demonstrated how feature selection impacted the results of the classifier. Statistical features proved to be ineffective for our model, as the results they produced weren't varied enough for our classifier to spot a trend in the data. Utilizing features like peak location, however, boosted the performance of our classifier. All in all, feature selection in machine learning is critical in the optimization process of classification. It is important to choose features well suited for a model, as they can improve or hinder performance in pattern recognition.

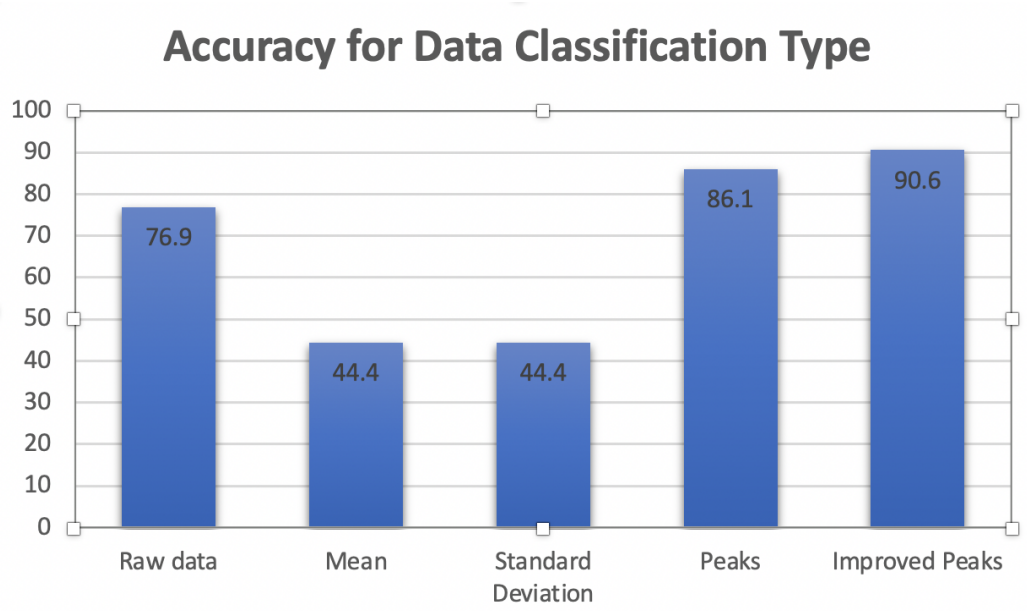


Figure 8: Accuracy Comparison Between the Different Classifications for Data, x axis depicts independent features, y axis depicts accuracy rate in percentages

4 FEATURE OPTIMIZATION OF GAUSS-NAIVE BAYES: PEAK LOCATION

Preliminary results from the Gauss-Naive Bayes classifier revealed that the training accuracy of peak location improved against features like mean and standard deviation, and had been comparable to the raw data classification for all of the walking, running, and jumping datasets. As a result, we chose to optimize peak finding through implementation of a numerical technique in order to see if it could improve the training accuracy further. In this section, we analyzed the Gauss-Newton method of a parabolic regression equation for isolated peaks, with some component of Caruana's algorithm to apply the quadratic model as a Gaussian fit, and a final peak finding algorithm to sort through our data and return our peak locations utilizing peak characteristic threshold limitations, which were lastly fed into the Gauss-Naive Bayes classifier for an improved classification output, as can be observed in Fig. 8 of the prior section.

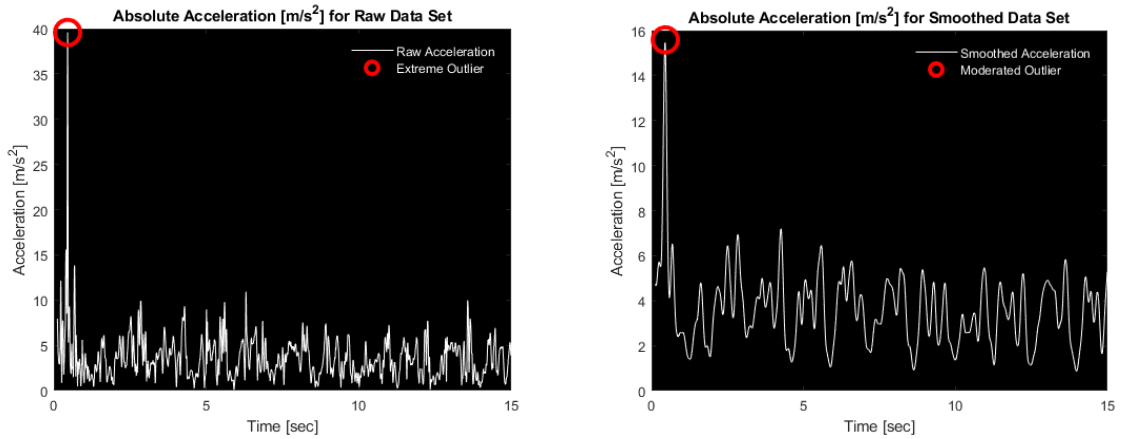
4.1 Defining Peaks in Our Dataset.

In statistical analysis, peaks in data sets are also called modes, local maximums, or global maximums, and they represent the high points on a graph. While they normally represent the distribution of a dependent variable in terms of quantity as typically used in probabilities or statistical modeling, the peaks in our graphs describe the mathematical trends of leg motion phases throughout performance of the three activities. Our dataset was a multimodal distribution, with peaks of varying acceleration amplitudes over the minute time duration for each dataset.

4.2 Noise Reduction with a Gaussian Smoothing Filter.

Noise in our experimental data emerged from extraneous vibrations picked up by the sensor as well as other random outliers including object obstructions in the path of the movement. From

previous findings done with the CNN classifier, we found that feature enhancement techniques such as applying a smooth filter, particularly Gaussian smoothing, over a dataset helped improve the accuracy of training. This is because smoothing mitigates the influence of outliers and noise while simultaneously revealing visible patterns across the data. In this situation, we applied a Gaussian smoothing filter across the acceleration values in each axial direction using a window threshold of 50, which smooths the data by application of a Gaussian over a fifty-element sliding window. Since there had been a lot of noisy peaks picked up by the sensor that did not affect our analysis, smoothing the data exhibited clearer and more detectable patterns to each major movement. The figure below (Fig. 9) displays the moderated effect of the extreme outlier after the smoothing filter, which had resulted from a misconstrued leg motion picked up by the accelerometer sensor. Since this outlier did not correspond to the trend of the other data points, it could be one of the sources of inaccuracy for the raw data classification performed earlier.



(a) (b)

Figure 9: Absolute Acceleration of Walking Dataset for Raw (a) and Smoothed (b) Acceleration

4.3 Numerical Methods: Applying Optimization Algorithms to Identify Peaks.

In our numerical analysis of the peaks, we first made the assumption that noise is random and normally distributed in our data set. With that, modeling the peaks and noise as a Gaussian makes sense because noise results from summing a large quantity of factors and distortions, which was smoothed to mitigate smaller obstructions to the data trend and reveal general patterns. Due to the relatively large quantity of peaks with noise in the data, we can apply the central limit theorem from probability theory to model the multimodal distribution throughout the time window with each mode distribution as a Gaussian. These assumptions from theory simplifications were how we approached our numerical analysis method in the following subsections.

4.3.1 Gauss-Newton Algorithm of a Quadratic Regression. The Gauss-Newton method is an iterative gradient-based optimization algorithm often used for nonlinear regression analysis. To compute a Gauss-Newton approximation, it is necessary to find the Jacobian matrix of partial derivatives in relation to the coefficients of the non-linear fit equation. For our peaks, we used the quadratic model approximation to implement as a Gaussian fit. Our algorithm scheme went as follows: we iterated through our sample data until the maximum iteration or until the tolerance was reached, solved for the partials in the Jacobian, and used a least-squares solution to compute

an array of the approximation coefficients. The MATLAB code in referral to the steps we took can be located as A-3 under the Appendices. The table below weighs in the advantages and pitfalls of this numerical approach, which is important to consider when using Gauss-Newton for other applications.

Table 1: Advantages and Pitfalls to the Gauss-Newton Algorithmic Approach

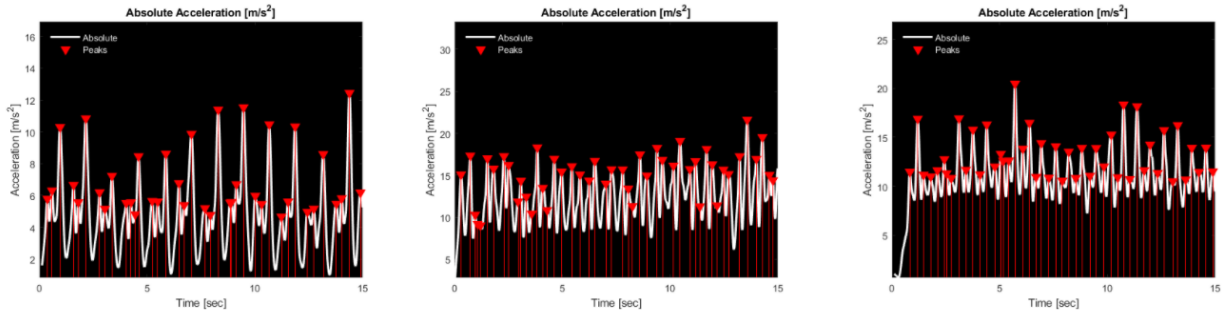
Advantages	Pitfalls
- Fairly efficient convergence rate for when step size and direction are determined - Fast and reliable processing speed for a small number of parameters - Works especially well near minimum or trough locations - Efficient computation without the second derivative aspect	- Convergence criteria heavily depends on good initial estimates and parameter size (may be slow or not find a solution at all without this) - Not suitable for design matrices that are ill-conditioned or deficient in rank - If residuals are large, the procedure will lose a lot of information

4.3.2 Caruana's Algorithm to Fit a Gaussian Function. Recalling that a Gaussian function is modeled by a mode's standard deviation and mean as parameters, we can use quadratic coefficients to relate to the Gaussian parameters. This is because by mathematical relation, a Gaussian function is the exponential of a quadratic equation. That being said, the coefficients for the quadratic function were obtained by the Gauss-Newton algorithm, then used to compute the Gaussian parameters, and finally implemented into the peak-finding algorithm to approximate the peak locations. A-4 in the Appendices illustrates the MATLAB code we used for this.

4.3.3 Finding Local Peaks algorithm. The last component to this numerical process is the local peak-finder algorithm, which identifies the local peaks from the smoothed dataset in all motions and their corresponding axial directions. The algorithm analyzes the dataset and appends the peak locations to an array if the criteria for a peak identification is met (zero-slope and maximum value over an amplitude threshold). This algorithm calls the function in the above subsection to output the x and y values of the peak location (the Gaussian modal distribution) when found.

4.4 Peak Location Results for a Sample Dataset of Walking, Running, and Jumping.

This section provides the sample precursive results for a randomly selected individual who participated in our experiment and their representative walking, running, and jumping absolute accelerations, respectively. Each sample motion depicts all of the peaks that were identified across our dataset, indicating their relative locations for a sampling time window of 15 seconds to better examine the trends. The peak finder also works for the x, y, and z- directionals. The following figure (Fig. 10) shows results for a randomly selected individual's walking, running, and time-series dependent accelerations as measured by the sensor and fed through our peak-finding algorithm. While the absolute acceleration peak trends are not identical across all the datasets for these motions, they are generally homogeneous across most of the sample individuals utilized in our datasets.



(a) (b) (c)

Figure 10: Peak Locations for Absolute Acceleration of (a) Walking, (b) Running, (c) Jumping

4.5 Discussion.

In performing analysis of our peak locations using numerical techniques to optimize the results, we noted some drawbacks and limitations to our approach. For one, there was a drawback with the loss of accuracy for narrower and sharper peaks in our approach of applying a Gaussian fit of the model, and especially when using raw data rather than smoothed data. This could be explicable in that sharper peaks are less representative of Gaussians. For characteristically narrower peaks, the normal distribution is heavily centered, with the x-location distribution less accurate due to the discrepancy in standard deviation away from the mode. Some possible methods that we could try in future implementations to counter the loss of accuracy of this method is through regularization techniques, like Gaussian Regularization or Tikhonov Regularization, to identify peaks with noise. Furthermore, we can see some future uses for this peak finder optimization in heart rate analysis, sound beat analysis, or any other motional gesture that can be described by periodic phases with noise occurring.

5 CONCLUSIONS

We saw reasonably good results with both machine learning methods we explored, with both having accuracies of around 90%. In terms of practical use, the Gauss Naive Bayes classifier with peak location optimization may prove more useful in time-pressured environments, like real time classification for fitness trackers, due to the much shorter training time needed for the classifier while the Deep CNN would be best used in environments where accuracy is a higher priority. Further improvements to our work would include expanding the classifier's training to larger, longer data sets with more diverse participants to improve accuracy and use cases. Adding more modes of movement, like skipping or jogging, to the motion detection would be useful as well. The work could also be expanded to include other machine learning methods and perhaps test the efficacy of the peak location optimization for other types of machine learning classifiers.

6 REFERENCES

[1] "Sensor Data Analytics (French Webinar Code)." *Sensor Data Analytics (French Webinar Code) - File Exchange - MATLAB Central*, www.mathworks.com/matlabcentral/fileexchange/54139-sensor-data-analytics-french-webinar-code.

[2] Myllymäki, Ossi. "Gauss-Newton Algorithm Implementation from Scratch." *Medium*, Medium, 28 June 2020, omyllymaki.medium.com/gauss-newton-algorithm-implementation-from-scratch-55ebe56aac2e.

[3] (toh@umd.edu), Tom O'Haver. *Peak Finding and Measurement*, terpconnect.umd.edu/~toh/spectrum/PeakFindingandMeasurement.htm.

[4] lopeLH. "LopeLH/Gaussian-Bayes-Classifier." *GitHub*, github.com/lopeLH/Gaussian-Bayes-classifier/blob/master/Gaussian%20Bayes%20classifier.ipynb.

[5] "Convolutional Neural Network." *Convolutional Neural Network - MATLAB & Simulink*, www.mathworks.com/discovery/convolutional-neural-network-matlab.html.

7 APPENDICES

A.1. GitHub link for reference source code that we each typed for our appropriate sections: https://github.com/f3lis/leg-motion-classification-ml

A.2. Google Drive link to the Raw Data files collected in Phyphox for our experiment: https://drive.google.com/drive/folders/1xV4FS2Rv4-W30WZVvx1ImVdhbVhcrPVc?usp=sharing

A.3. Deep Convolutional Neural Network Code:

```
function [net, info, test_acc] = CNN(train_data, train_labels, test_data, test_labels)
    width = size(train_data, 1); % should be 500
    channels = size(train_data, 2);
    N = size(train_data, 3);
    train_data = reshape(train_data, [1, width, channels, N]);
    N_t = size(test_data, 3);
    test_data = reshape(test_data, [1, width, channels, N_t]);
    % Hyper-parameters
    c1_num = 10; % conv 1 number of filters
    c1_len = 30; % conv 1 filter length
    p1_sz = 15; % max pool 1 size
    p2_sz = 10; % max pool 2 size
    c2_num = 20; % conv 2 number of filters
    c2_len = 10; % conv 2 filter length
    % Network
    input = imageInputLayer([1 width channels]);

    c1 = convolution2dLayer([1 c1_len], c1_num,'stride',1);
    b1 = batchNormalizationLayer;
    p1 = maxPooling2dLayer([1 p1_sz],'stride', p1_sz);
    c2 = convolution2dLayer([1 c2_len], 2 * c2_num,'stride',1);
    b2 = batchNormalizationLayer;
    p2 = maxPooling2dLayer([1 p2_sz],'stride', p2_sz);
    f1 = fullyConnectedLayer(20);
    f2 = fullyConnectedLayer(3);
    s1 = softmaxLayer;
    output = classificationLayer;
    convnet = [input; c1; b1; p1; c2; b2; p2; f1; f2; s1; output];
    % Cross Validation
    opts = trainingOptions('sgdm', 'MaxEpochs', 60, ...
           'ValidationData', {test_data,test_labels},  …
```

```
            'ValidationFrequency', 1, 'Verbose',false);
    [net,info] = trainNetwork(train_data, train_labels, convnet, opts);
    test_pred = classify(net, test_data);
    test_acc = sum(test_pred == test_labels) / numel(test_labels);
end
```

A.4. Gauss-Newton Code:

```
function a = gaussnewt(t,acc)
% Application of Gauss-Newton method with a polynomial of degree 2
% of a quadratic fit to apply as a Gaussian
% Inputs: time, acceleration data
% Outputs: Array a enlisting coefficient values for fit equation
tol = 0.0001;
imax = 50;
n = length(t);
a = [-0.001,-0.001,0.001]; % Approximate starting values
for i = 1:imax
      a0 = a(1);
      a1 = a(2);
      a2 = a(3);
      for i = 1:n
      % Quadratic function being used as a model for regression
      h(i) = a0.*t(i)^2 + a1.*t(i) + a2;
      dh_a0 = t(i)^2;
      dh_a1 = t(i);
      dh_a2 = 1;
      J(i,1) = dh_a0; % Jacobian coeff 1
      J(i,2) = dh_a1; % Jacobian coeff 2
      J(i,3) = dh_a2; % Jacobian coeff 3
      d(i) = acc(i) - h(i); % Residual matrix
      end
      ea = (J'*J)\(J'*d'); % Least squares solution
      a = a + ea';
      output = [i a ea'];
      if (abs(ea(1)) < tol && abs(ea(2)) < tol) % Check approximate error
            disp('Gauss-Newton method converged.');
            break
      end
      end
end
```

A.5. Caruana's Algorithm for fitting Quadratic to Gaussian:

```
function [y_loc,x_loc,width]=gaussfit(x,y)
% Gaussfit outputs the y,x,width of the gaussian from a quadratic fit
% This is an application of Caruana's algorithm which relates
% quadratic to Gaussian using log scaled data
% Input: x,y data
% Output: y,x,width of Gaussian
maxy = max(y);
for p = 1:length(y)
      % Prevent errors from negative log
      if y(p)<(maxy/1000)
            y(p)=maxy/1000;
      end
end
logy_scale=log(abs(y));
coef = gaussnewt(x,logy_scale);
c = coef(3);b = coef(2);a = coef(1);
mu = mean(x);
sigma = std(x);
% Compute peak position and height of the fitted parabola
```

```
x_loc = -((sigma.*b/(2*a))-mu);
y_loc = exp(c-a*(b/(2*a))^2);
width = norm(sigma.*2.35703/(sqrt(2)*sqrt(-1*a)));
end
```