December 9, 2020

Activity Recognition in Healthy Older People
Project Phase III

Jordan Lanius
CS 4300
University of Missouri-St. Louis

Abstract

As she has grown older, my grandmother has experienced dizziness and infrequent fainting spells. She has a button that she wears around her neck that she can use to call for help in case of a fall, but it is risky to rely on her ability to press it during an emergency. I am interested in the potential for an artificial intelligence to monitor a person's activity and be able to report any concerning behavior.

1 Dataset

The dataset was obtained by the UCI Machine Learning Repository and is titled "Activity recognition with healthy older people using a batteryless wearable sensor Data Set"[1]. It was provided by Roberto Luis Shinmoto Torres, Damith Ranasinghe, and Renuka Visvanathan from the University of Adelaide. For this project, I am only using the S1 setting that uses four RFID reader antennas. There are sixty participants that took part in this trial, with varying amounts of data collected from each.

1.1 Dataset Description

The data was originally separated into different files for each participant. During the cleaning process, where I merged it from multiple files to a single *.csv file, I included columns for the participant number and their gender. I also adjusted the activity column, condensing it from four different activities into two categories: 'In Bed', expressed as 0, and 'Out of Bed', expressed as 1.

In total, the condensed dataset has 52482 rows and 11 columns. For this project, I will be using 8 of the 11 columns for the input and use the 'Activity' column for the output. The input attributes that will be used are:

- Gender of the Participant
- Acceleration reading in G for frontal axis
- Acceleration reading in G for vertical axis
- Acceleration reading in G for lateral axis
- ID of antenna reading sensor
- Received signal strength indicator (RSSI)
- Phase
- Frequency

1.2 Input Data Visualization

Histograms plot the distribution of each input feature. **Figure 1** shows these plots for each input column of the dataset.

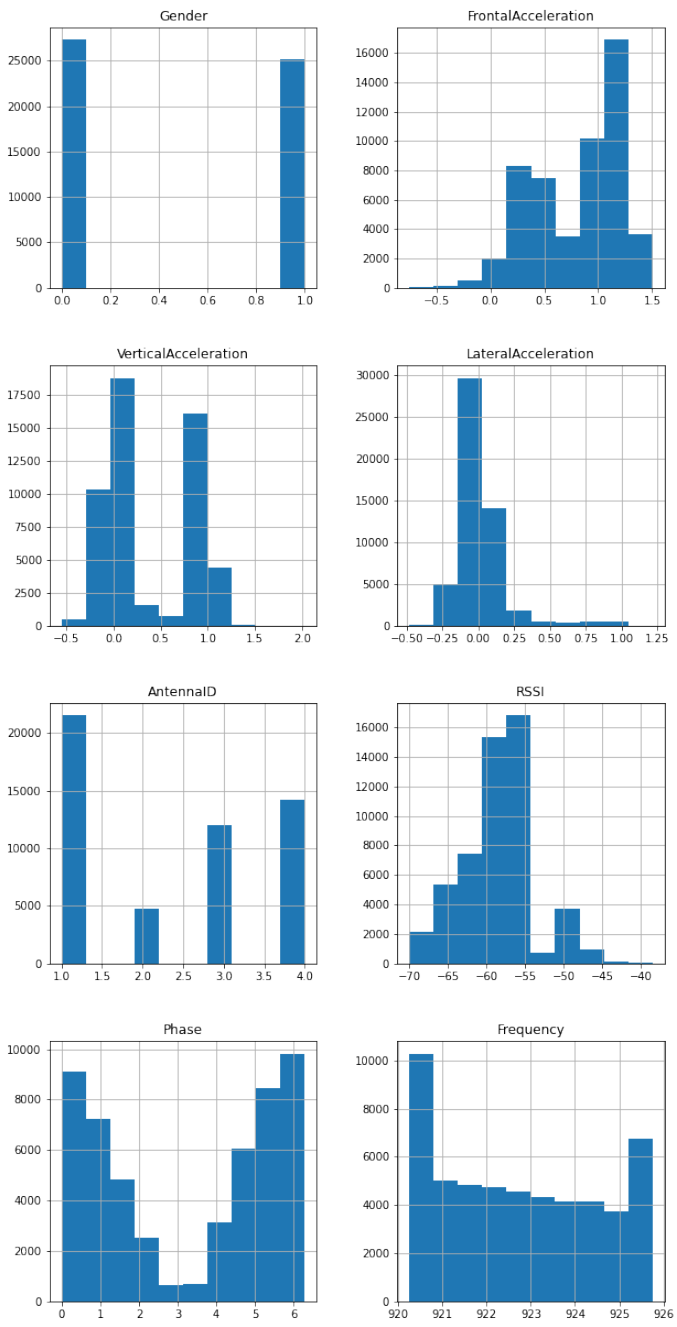

Figure 1: Input Data Distribution Histograms

Table 1 contains the minimum and maximum values of each attribute as well as the calculated mean and standard deviation values that are used for normalization.

	Min	Max	Mean	Std
Gender	0.0	1.0	0.479993	0.499604
Frontal Acceleration	-0.74808	1.5032	0.805042	0.39636
Vertical Acceleration	-0.55349	2.0302	0.377804	0.468899
Lateral Acceleration	-0.48121	1.2178	0.00771	0.180674
Antenna ID	1.0	4.0	2.360752	1.261542
RSSI	-70.0	-38.5	-58.430814	4.61122
Phase	0.0	6.2817	3.275907	2.240341
Frequency	920.25	925.75	922.762261	1.693769

Table 1: Input Feature Statistics

1.3 Output Data Visualization

Figure 2 shows the distribution of the binary output. There is some imbalance between the two. About 87.9% (46145/52482) of the sensor observations are classified as 'In Bed' while only 12.1% (6337/52482) are classified as 'Out of Bed'. While significant, this imbalance isn't extreme and there are still plenty of observations of both classes.

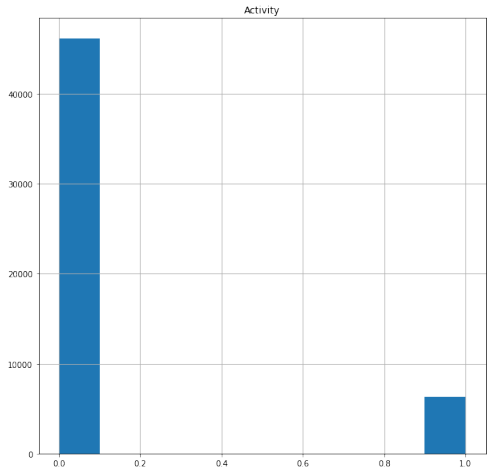

Figure 2: Output Data Distribution Histogram

2 Data Processing

2.1 Data Splitting

The data has been shuffled randomly and separated into two groups. At first, 70% of the dataset was allocated as training data and the remaining 30% of the dataset has been reserved as validation data. Beginning in **Section 3.2** this was changed to 80% allocated as training data and 20% reserved as validation data.

The validation data will be used to evaluate the different architectures explored in **Section 3**. A model isn't effective if it is only accurate with the training data.

2.2 Data Normalization

Neural networks work best when data has been distributed uniformly. The best way to achieve this is to apply a normalization technique to the dataset. For this project I have used Standardization which utilizes **Equation 1**.

$$X = (X - mean)/std \tag{1}$$

3 Modelling

3.1 Model Architecture

The most important part of constructing a neural network is the architecture of the model. There are two functions that are commonly used in the final layer of the network. These are the Sigmoid and Linear activation functions. Sigmoid is used primarily for classification problems, like this one, and Linear is used for linear regression problems. Both will be explored in this section.

3.1.1 Sigmoid Activation

Table 2, given below, compares the performance statistics for five different architectures that use the sigmoid activation function. The models labeled 'Relu' use the rectified linear activation function on all but the final layer of neurons. The models labeled 'Pure' use the sigmoid activation function on every layer of neurons.

	Accuracy	Precision	Recall	F1 Score	Validation Accuracy
Single-Layer	91.62%	82.50%	39.26%	0.53	91.65%
Relu Double-Layer	97.76%	97.44%	83.70%	0.90	97.85%
Pure Double-Layer	97.15%	99.97%	76.54%	0.87	97.27%
Relu Triple-Layer	98.29%	97.75%	87.95%	0.93	98.15%
Pure Triple-Layer	98.23%	98.82%	86.42%	0.94	98.39%
Relu Quadruple-Layer	98.56%	98.47%	89.49%	0.94	98.39%
Pure Quadruple-Layer	98.34%	98.17%	87.95%	0.93	98.34%

Table 2: Performance comparison for various sigmoid activation architectures

As seen above, all of the neural networks performed better than a basic logistic regression model. Between the double-layer models, the one that used the linear activation function on every layer performed the best. There is no significant difference between the triple-layer models, although the one that only used the sigmoid activation function performed a bit better with the validation data. Neither of the quadruple-layer models were able to improve the accuracy in any meaningful way and any further growth risks overfitting. I will continue this project using a three-layered sigmoid model, calling it the ***Original Neural Network***.

3.1.2 Linear Activation

Table 3, given below, compares the performance statistics for five different architectures that use the linear activation function. The models labeled 'Relu' use the rectified linear activation function on all but the final layer of neurons. The models labeled 'Pure' use the linear activation function on every layer of neurons.

	Accuracy	Precision	Recall	F1 Score	Validation Accuracy
Single-Layer	87.94%	90.62%	0.65%	0.01	88.07%
Relu Double-Layer	97.43%	94.91%	83.25%	0.89	97.55%
Pure Double-Layer	87.92%	86.36%	0.43%	0.01	88.05%
Relu Triple-Layer	98.14%	97.92%	86.55%	0.92	98.10%
Pure Triple-Layer	87.97%	93.02%	0.90%	0.02	88.10%

Table 3: Performance comparison for various linear activation architectures

Overall, using the linear activation function didn't produce any significantly better results than using the sigmoid activation function. The models that performed the best utilized the rectified linear activation function for their input and hidden layers. In fact, using the linear activation function for every layer led to worse performances.

3.2 Hyperparameters

In addition to the model's architecture, there are other parameters of the training process that can be manipulated. I explored different variations of neuron numbers, optimization algorithms, batch sizes, and number of epochs. **Table 4** contains the performance statistics of the most notable sets of hyperparameters.

	Accuracy	**Precision**	**Recall**	**F1 Score**	**Validation Accuracy**
RMSprop 16 Neurons 1024 Epochs Batches of 16	98.53%	97.21%	90.43%	0.94	98.69%
RMSprop 16 Neurons 1024 Epochs Batches of 64	98.62%	98.18%	90.21%	0.94	98.66%
SGD 16 Neurons 1024 Epochs Batches of 8	98.26%	98.10%	87.25%	0.92	98.48%
Nadam 16 Neurons 256 Epochs Batches of 8	98.81%	99.00%	91.07%	0.95	98.71%
Nadam 16 Neurons 1024 Epochs Batches of 8	99.13%	98.94%	93.83%	0.96	98.84%

Table 4: Performance comparison for various 3-layer, sigmoid activation models

Because of it produces a better training and validation accuracy, along with the other statistics, I will continue this project using the last set of hyperparameters in **Table 4**. This model will be called the ***Optimized Neural Network***.

3.3 Feature Importance and Removal

Having more attributes available generally leads to a more accurate algorithm. However, not every feature has the same importance. This section attempts to clean the dataset of attributes that don't contribute significantly to the algorithm's accuracy.

3.3.1 Feature Importance

I wrote a function that cycled through every attribute and train the model with only that attribute. The accuracy generated by each solo feature will be used in **Section 3.3.2** to establish an order for removal. The function also displayed the resulting accuracy values in **Table 6** and **Figure 3**.

Isolated Feature	Accuracy
Gender	88.07%
FrontalAcceleration	88.24%
VerticalAcceleration	89.47%
LateralAcceleration	88.07%
AntennaID	96.93%
RSSI	88.50%
Phase	88.07%
Frequency	88.07%

Table 5: Individual Feature Accuracy

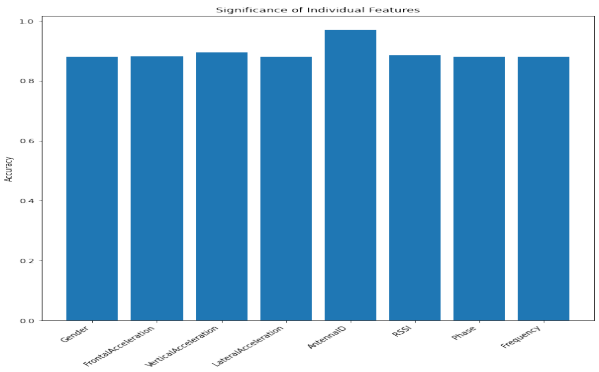

Figure 3: Significance of Individual Features

3.3.2 Feature Removal

I ordered the attributes in an ascending order based on the accuracy they produced in **Section 3.3.1**. I then passed this list into a function that removed the least important feature from the dataset before training the model. The function produced **Table 6** and **Figure 4** which shows how quickly the accuracy of the model falls with each removal.

Features	Accuracy
All Features	98.64%
Gender Removed	98.48%
LateralAcceleration Removed	98.67%
Phase Removed	98.53%
Frequency Removed	98.34%
FrontalAcceleration Removed	98.34%
RSSI Removed	96.96%
VerticalAcceleration Removed	96.92%

Table 6: Accuracy After Removing Feature

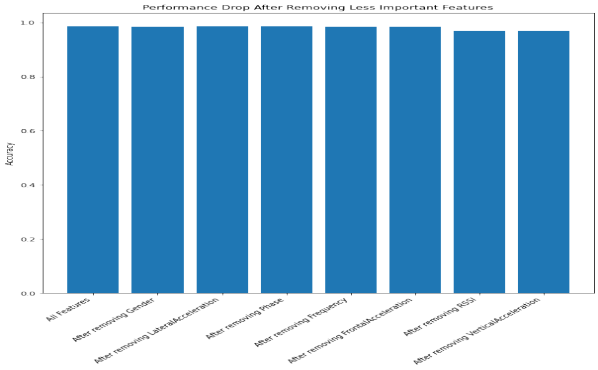

Figure 4: Performance After Removing Least Important Features

According to the resulting data, a very accurate prediction can be found from new data even when the Gender, LateralAcceleration, Phase, Frequency, and FrontalAcceleration features are ignored. From now ow, the model using a dataset consisting of only AntennaID, VerticalAcceleration, and RSSI will be called the ***Feature-Reduced Neural Network***.

4 Model Evaluation

4.1 Performance Comparison

Table 7, given below, compares the performance statistics for the three major architectures explored above. It's purpose is to establish the effectiveness of each variation of the selected neural network to the simple logistic and linear regressions.

	Accuracy	Precision	Recall	F1 Score	Validation Accuracy
Logistic Regression	91.62%	82.50%	39.26%	0.53	91.65%
Linear Regression	87.94%	90.62%	0.65%	0.01	88.07%
Original NN	98.23%	98.82%	86.42%	0.94	98.39%
Optimized NN	99.13%	98.94%	93.83%	0.96	98.84%
Feature-Reduced NN	98.26%	97.99%	87.46%	0.92	98.25%

Table 7: Performance comparison for selected architectures

The Linear Regression is the worst of the five. This is most likely due to the fact that the linear algorithm was not designed to solve classification problems like this project. On the other hand, all of the other models use the sigmoid algorithm, which works very well for classification. Even the simple Logistic Regression has an accuracy of over 90%.

Improving the architecture of the model and the hyperparameters of the training each increased the effectiveness of the algorithm. They are now so precise that there is little room for further improvement. However, they are still not perfect.

Removing the least important attributes from the dataset slightly reduced the accuracy of the algorithm. However, this change was not significant as the result was still on par the the Original Neural Network.

4.2 Learning Curves

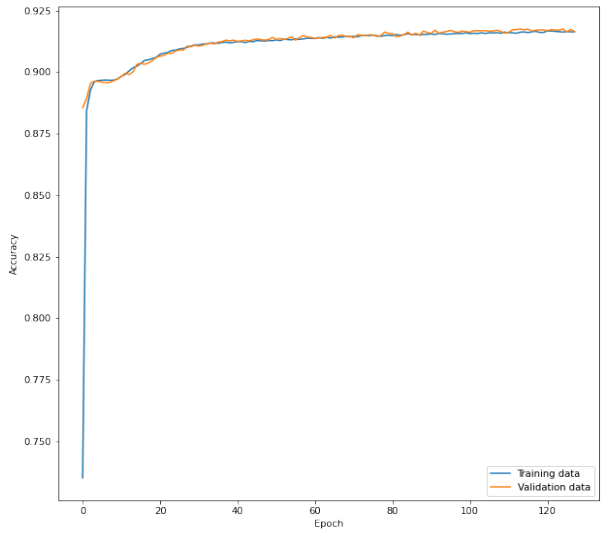

Figure 5: Logistic Regression Learning Curve

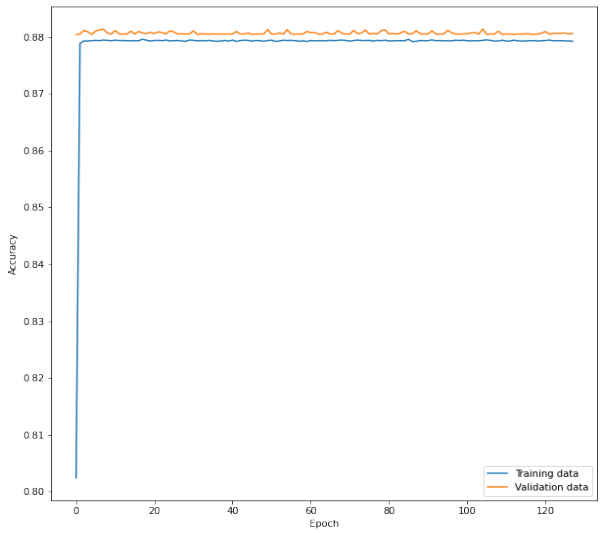

Figure 6: Linear Regression Learning Curve

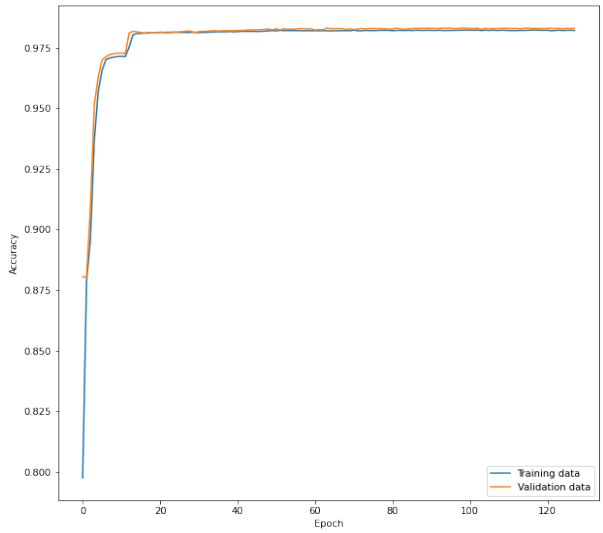

Figure 7: Original Neural Network Learning Curve

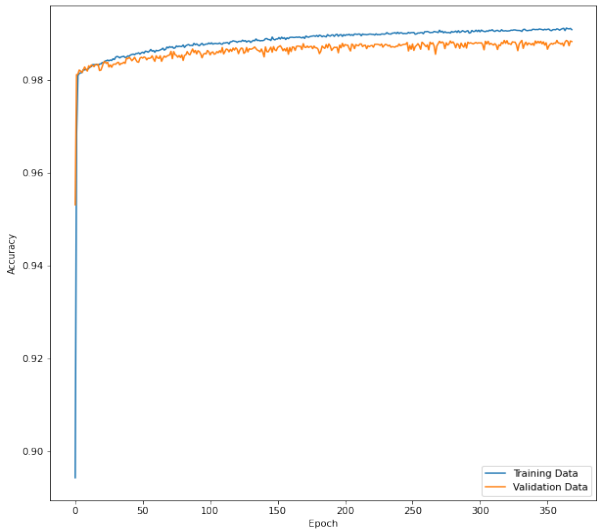

Figure 8: Optimized Neural Network Learning Curve

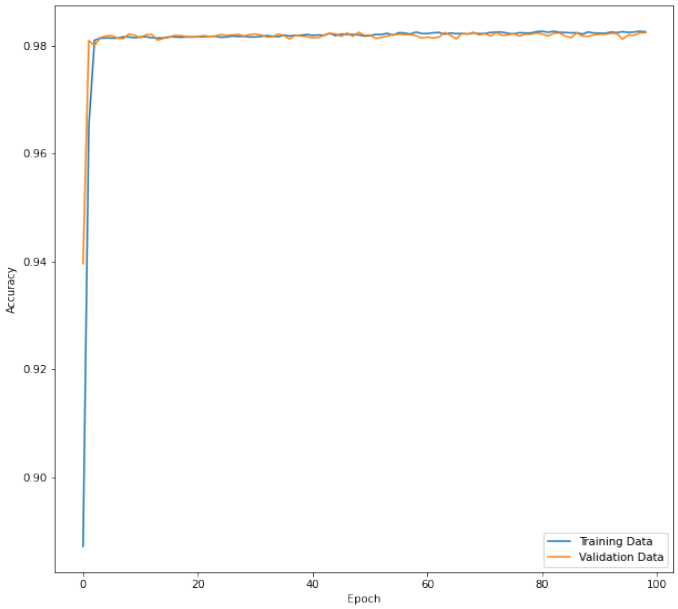

Figure 9: Feature-Reduced Neural Network Learning Curve

All five figures show very clean learning curves. The logistic regression shown in **Figure 5** has a gradual slope compared to the others. The linear regression shown in **Figure 6** almost immediately attains its max accuracy and barely has any room to improve. The neural network shown in **Figure 7** quickly attains a very high accuracy after only a few epochs and very slowly continues to rise. When optimized, the model shown in **Figure 8** becomes a less extreme slope than the Original Neural Network, yet it continues improving at a faster rate. The completed Feature-Reduced Neural Network's learning curve shown in **Figure 9** is closer to the Original Neural Network's learning curve in shape, quickly reaching a plateau. While it points to slower growth in the future, it manages to reach an accuracy similar to the Optimized Neural Network.

5 Conclusion

This phase of the project focused on continuing to fine-tune our model to better understand the data. A simple architecture was built upon with carefully selected hyperparameters to optimize the accuracy of the algorithm. The development of this algorithm is the ultimate purpose of this project. Similarly, the dataset was trimmed down to only the attributes that have significant impacts on the results.

Experimentation has resulted in the selection of a triple-layer model, with 16 neurons in the upper layers, and uses the sigmoid activation function on all levels. It optimizes itself with the Nadam algorithm and runs for a maximum of 1024 epochs in batches of 8. It provides the highest accuracy of all models tested so far.

Most of the attributes from the dataset are able to be ignored before any significant drop in prediction accuracy. The most significant features are AntennaID, VerticalAcceleration, and RSSI.

References

[1] Shinmoto Torres, R. L., Ranasinghe, D. C., Shi, Q., Sample, A. P. (2013, April). Sensor enabled wearable RFID technology for mitigating the risk of falls near beds. In 2013 IEEE International Conference on RFID (pp. 191-198). IEEE.