Final Report of Physical Therapy

Yongrong Chai, Keliang Xu, Jack Carbaugh

5/9/2022

Abstract

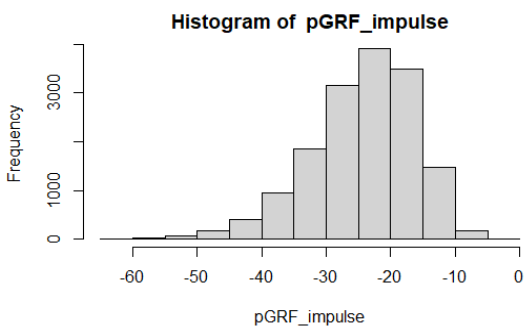
This study is an analysis of a lot of time series and discrete data from client's studying osteoporosis of the knee. The purpose of this report is to provide statistical analysis to this study with the main goal of analyzing whether there are any noticeable patterns in the data, and whether we can use dimension reduction to locate the most important elements. We did some exploratory data analysis to see the distribution and line shape of the time series and discrete data. We used PCA(principal component analysis) to give the important factors and used Autoencoder to reduce dimensionality as well as search for anomalies.

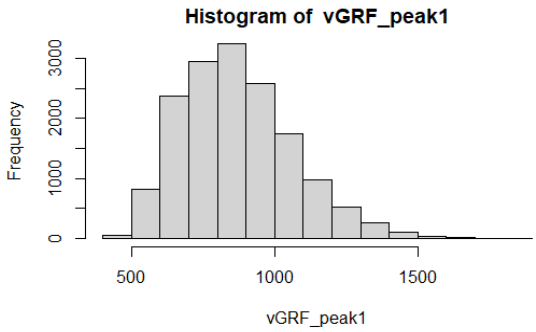
Introduction

Our client is Kerry Costello from BU SAR Movement and Applied Imaging Lab. The client is studying osteoporosis of the knee by imaging subjects' walking motion over a series of pressure plates. This process produces a lot of time series and discrete data, and the client would like to know if there are any noticeable patterns in the data, if we can use dimension reduction to locate the most important elements.

Exploratory Data Analysis

We first made some histograms of the discrete data. Most variables appear to be normally distributed, with some being slightly skewed. Some would likely need to be log transformed, such as the bottom left variable in figure 1, while others are in percentages, and may need to be logit transformed.





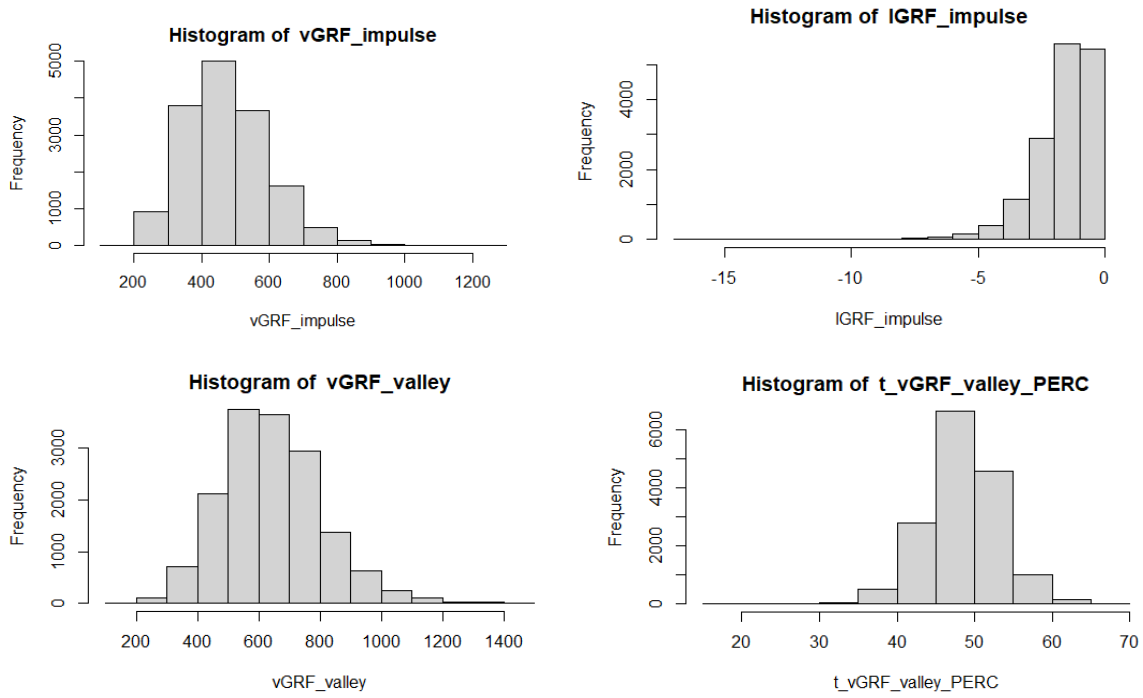


Figure 1: Histograms of the discrete data

We also tried to set up some scatter plots of the discrete data to see if there were any clear relationships. Not many showed this though, with only a few linear relationships showing up between the similar peaks across the walk cycle.

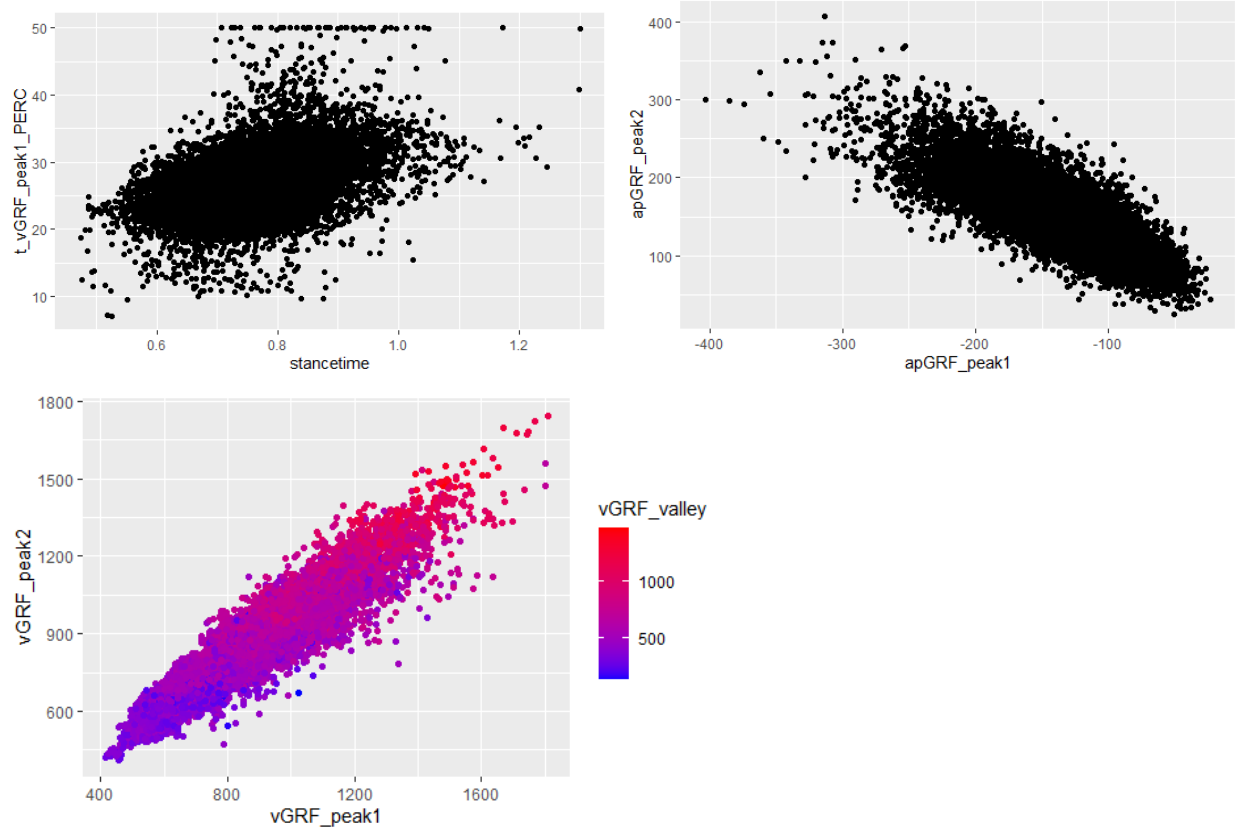


Figure 2: Scatter Plots of Discrete Data

The time series plots mainly lie force and moment in three dimensions x, y, z. In the first line of plots is force in x, y, z. As we can see, GRFx and GRFz have roughly the same process, while GRFy is more irregular. For the observation of the moment, we can only draw a similar range of positive and negative values. Therefore, we need to do a more in-depth understanding of these datasets in the next step.

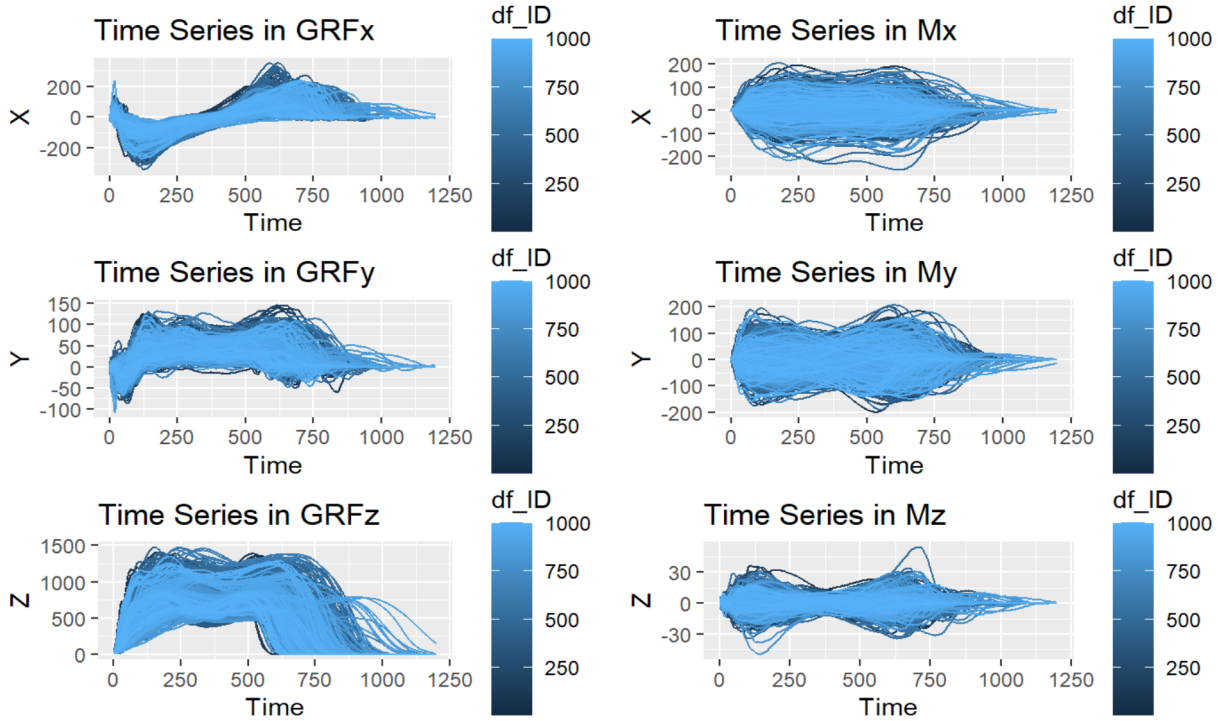
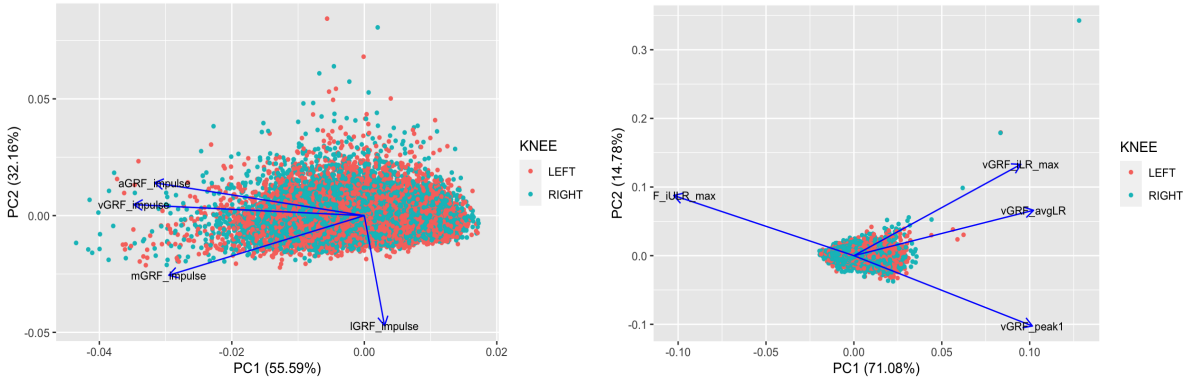


Figure 3: Time series data as line plots

Modeling

Principal Component Analysis

Based on the discrete file, we splitted the variables into 4 parts based on 4 different units. So there are 4 biplots here, it displays both the principal component score and the principal component loadings. In these plots, the longer the arrow, the greater its weight.



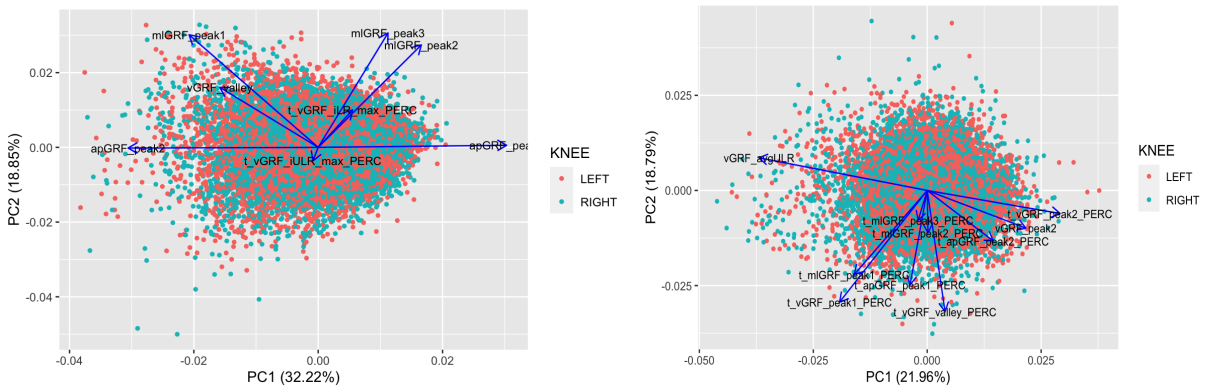


Figure 4: PCA Visualization

We are interested to know what are the important factors that emerge from the PCA (i.e. which ones explain a lot of variation). The second row, Proportion of Variance, shows how much variation in the data is described by each component; Let's take a look at the first one, which was high-lighted in blue, we will focus on the first 2 principal components, PC1 & PC2, which together explained 87.76% of the observed variation.

```
pca.impulse.summary$importance #first2 - 0.555930+0.321630
pca.unloading.summary$importance #first2 - 0.710820+0.1478000
pca.peak.summary$importance #first5 - 0.322190+0.188500+0.137390+0.1226800+0.0957700
pca.time.summary$importance #first6
```

```
                          PC1      PC2       PC3       PC4
Standard deviation     1.491218 1.134256 0.5719785 0.4032038
Proportion of Variance 0.555930 0.321630 0.0817900 0.0406400
Cumulative Proportion  0.555930 0.877570 0.9593600 1.0000000
```

Figure 5: Interesting Finds in PCA by R

Here are three different ways to visualize top important features which together explained more than 80% of the observed variation. Plot 2 shows top 5 features specifically.

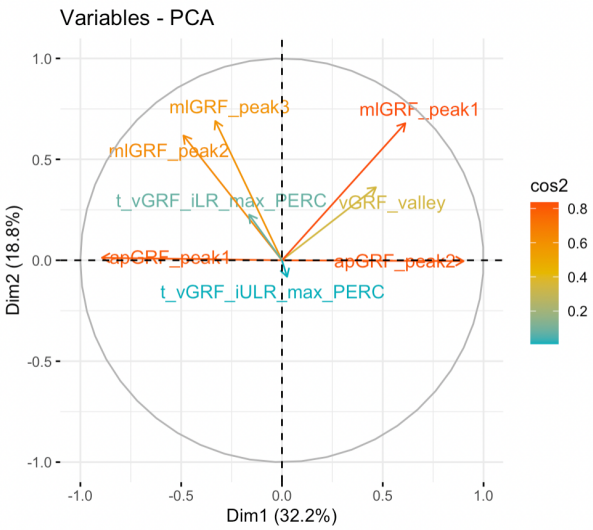
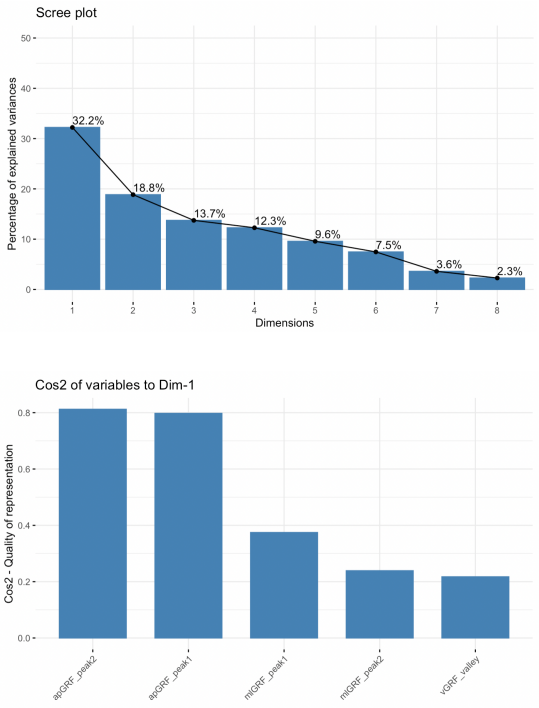


Figure 6:Another PCA Visualization



Baseline Model - Functional PCA

For our baseline model, we tried using functional PCA for dimension reduction. We first attempted it on the different GRF time series. The alignment process was slow, so our current results only include the first 20 samples. From these plots, we mainly see that the PCs match up with the line plots from our EDA.

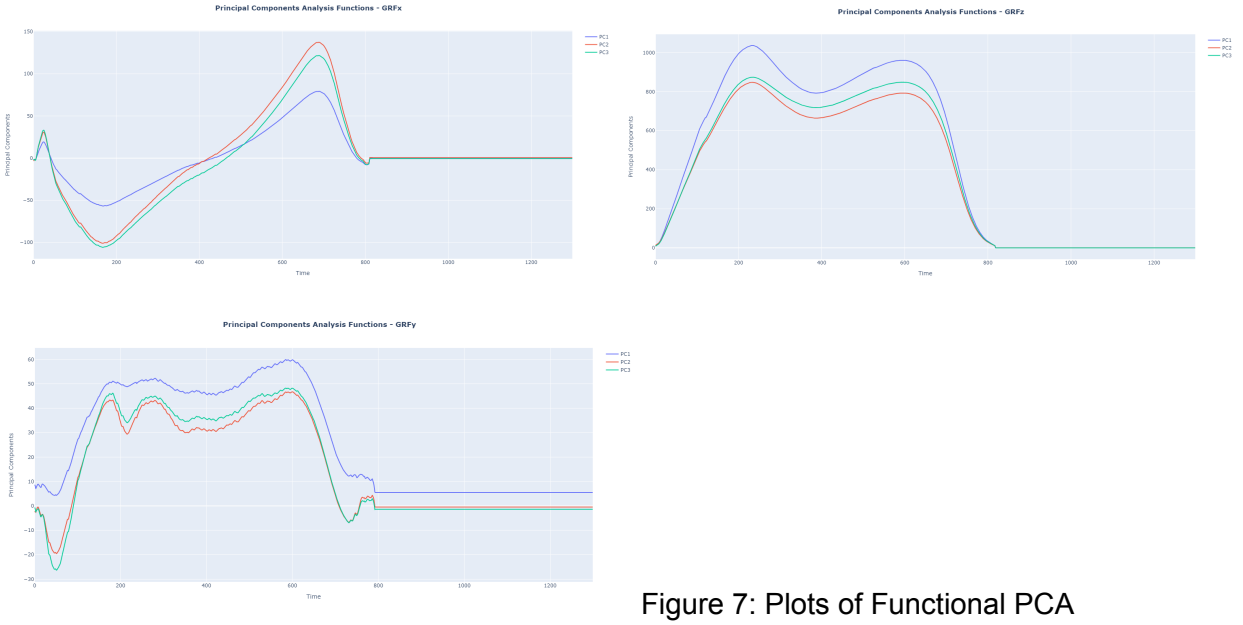


Figure 7: Plots of Functional PCA

We then plotted each of the observations based on the first two principal components. First plot is the result for GRFx. Here, we see a pretty big distinction between the first PC of subject 1 and 2 from subject 3. Second plot is the result for GRFy. Now, all three subjects have their own distinct clusters, with a noticeable distinction in PC2 between subject 3's right and left knees. Third plot is the result for GRFz. Just like for GRFy, there's clear distinctions between each of the different subjects.

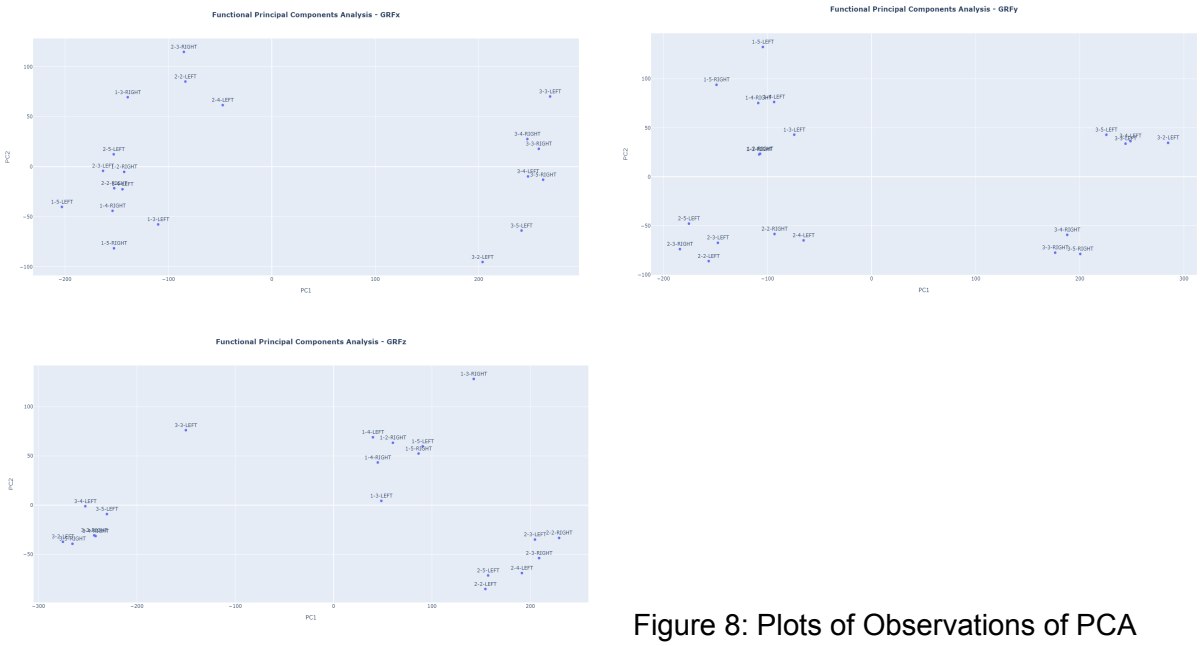


Figure 8: Plots of Observations of PCA

Neural Network - Autoencoder

Before setting up the model, we needed to clean up the data. We focused on the 8 non-time-normalized sequence variables, as this provided the model with much more data. The data was cleaned by adding on the indices, and removing all data past time point 1300, as after this point every variable of every sample was 0. All 8 variables were then transformed into numpy arrays and combined into one three dimensional array. Finally, each variable was scaled to be within the range 0 to 1, to avoid bias towards better reconstruction for high magnitude variables.

The model we used for this analysis was a vanilla autoencoder. An autoencoder is a type of neural network with the purpose of recreating the input data after dimension reduction. The autoencoder does this by encoding the data through a bottleneck, which greatly reduces the dimensionality of the data. The data can then be decoded from the bottleneck, and expanded to the original input size. An autoencoder is working well if the reconstructed data emulates the input data, and the general patterns of the data are captured.

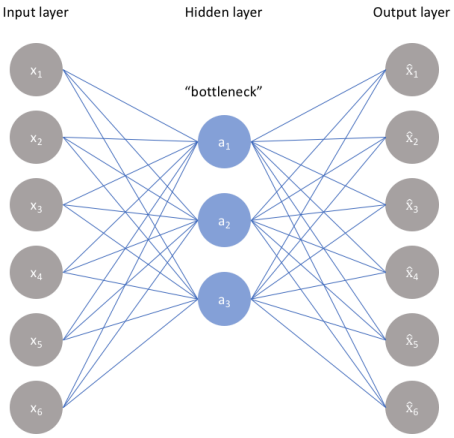


Figure 9: The basic Autoencoder structure

The autoencoder was built using the Keras package. One hidden layer was used, which was 100 variables. This means that the autoencoder reduced the data from 10,400 variables down to 100 through the bottleneck. Leaky ReLu activation was found to be the most effective activation function for this model. 50 epochs were used to train. This model produced highly accurate reconstructions, with most attempts having MSE loss between 3e-4 and 5.5e-4. An example of the reconstruction results can be seen in figure 10.

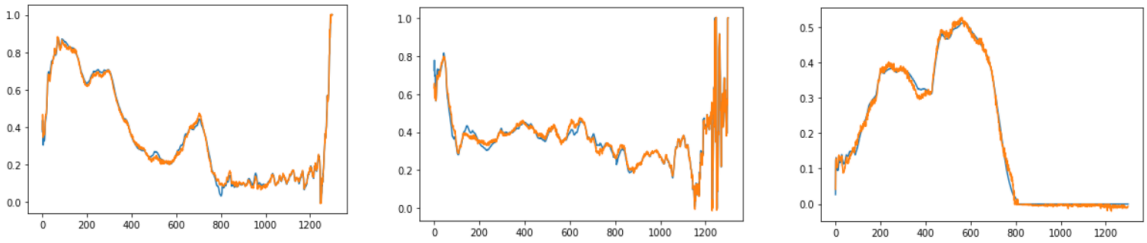


Figure 10: Singular sample autoencoder reconstruction results for GRFx, GRFy, GRFz, respectively. (Blue = actual, Orange = reconstructed)

One application of the autoencoder beyond dimension reduction and reconstruction is anomaly detection. If the autoencoder has trouble recreating a certain sample, this may indicate that the sample is an anomaly and deviates significantly from the average sequence. There are a few methods of determining anomaly, but our group went with the simplest: observing which reconstructions had the highest MAE error. The boundary can be chosen algorithmically or

directly. For simplicity, we directly picked the boundary, usually between 0.03 and 0.04, depending on the amount of model MSE error. This boundary would usually give between 200 and 300 anomaly samples.

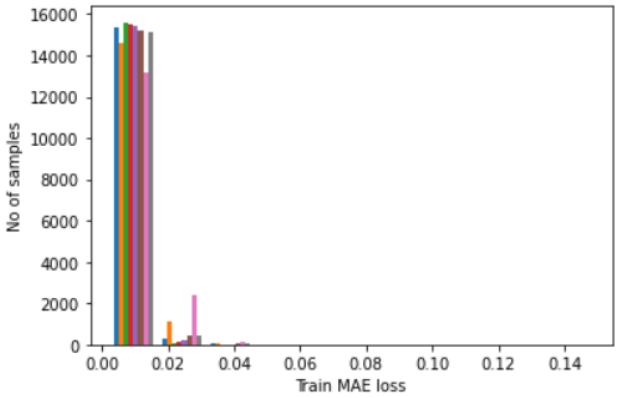


Figure 11: One example of Histogram showing training error. Each color is a different variable.

As neural networks include random elements, the anomalies detected will differ between model iterations. However, we noticed that some samples repeatedly appeared as anomalies despite this. This would include samples 797, 798, 800 which relate to subject 123, samples 1803-1806 which relate to subject 273, and samples 3789-3792 which relate to subject 572. These subjects in particular may have had either bad sampling, or have the tested knee condition. The boundary for loss can be lowered further to potentially predict afflicted knees.

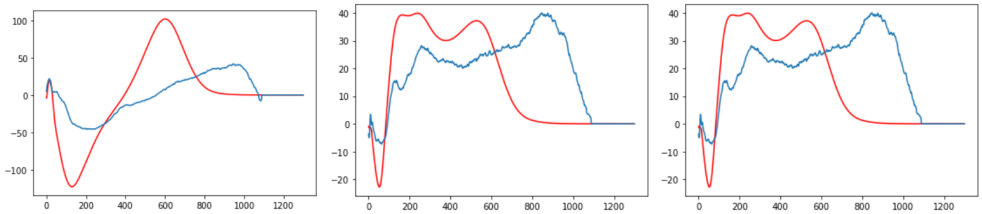


Figure 12: Sample 797 (Blue) against the average sample (Red) for GRFx, GRFy, and GRFz, respectively. Sample 797 has a noticeably different trend than the average.

Conclusion

Important Factors according to PCA

Here are the import factors which were found according to PCA.

vGRF_avgLR	vGRF_impulse
vGRF_iLR_max	mGRF_impulse
vGRF_avgULR	vGRF_valley
vGRF_peak2	apGRF_peak1
t_vGRF_peak1_PERC	apGRF_peak2
t_vGRF_peak2_PERC	mlGRF_peak1

t_vGRF_valley_PERC	mlGRF_peak2
t_apGRF_peak1_PERC	

Table 1: Important Factors

Autoencoder

Vanilla Autoencoder used on multivariate time series for GRFx, GRFy, GRFz.
Keras - One layer used; Leaky Relu activation. Dimensionality reduced down to 100 variables.
Sample Reconstruction results: (Blue = actual, Orange = autoencoder result)

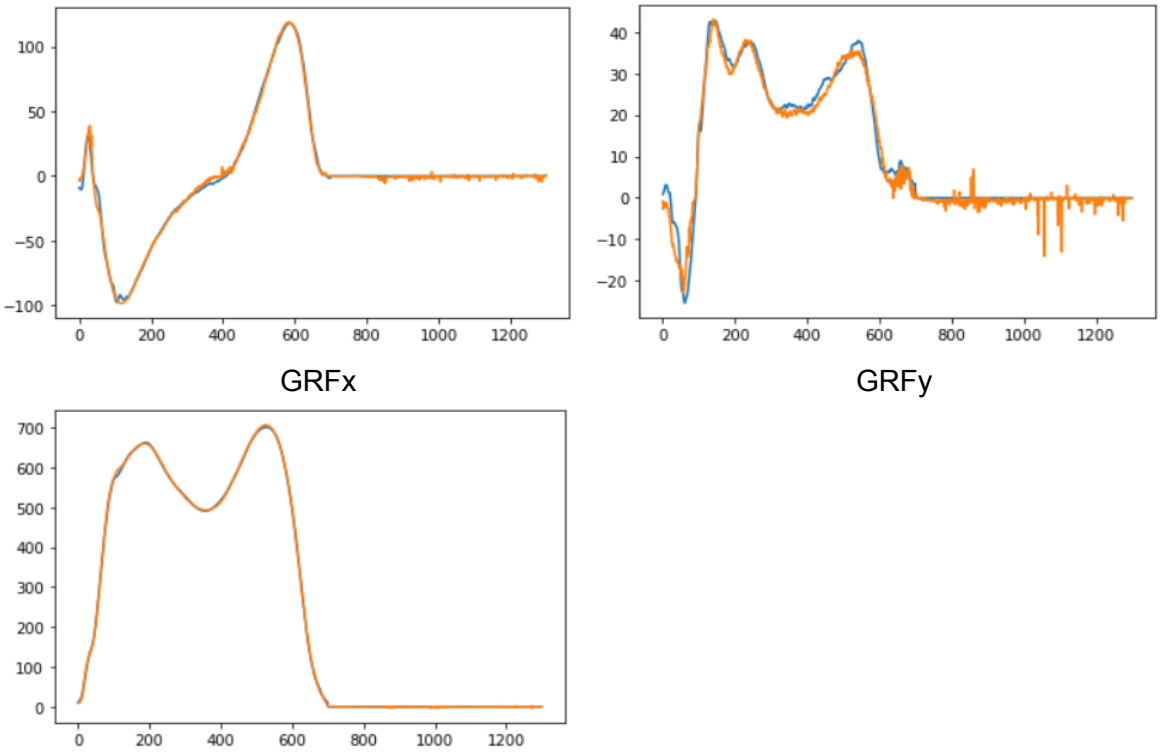


Figure 13: Sample Reconstruction Results

Citation

Image source: [GitHub - shobrook/sequitur: Library of autoencoders for sequential data]

Appendix

CH Index

Here is another method for choosing K which is the number of clusters. We can get the k by looking at the ch index plot to find the signal of when that happens the local maximum measure (the measure will drop and rise again) should be our ideal cluster number. However, in this graph, it is hard to detect k.

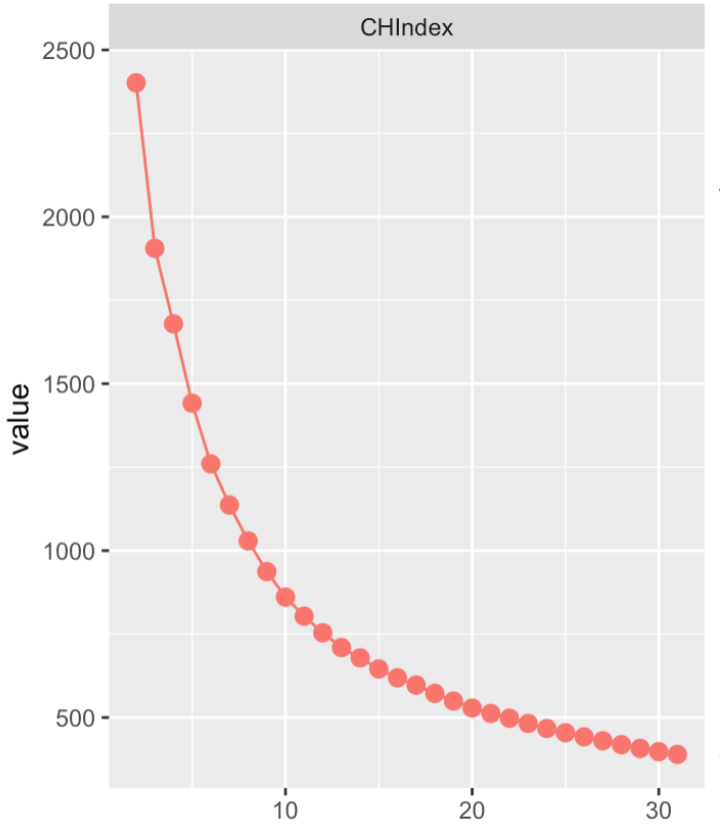


Figure 14: Result of CH Index

Time Series Forecasting Methods – ARIMA Model

We found the time series forecasting methods and tried it.

To estimate the trend component and seasonal component of a seasonal time series that can be described using an additive model, we can use the “decompose()” function in R. This function estimates the trend, seasonal, and irregular components of a time series that can be described using an additive model.

In this case, we use frequency = 10 which allows us to reduce the amount of data tenfold. As we can see in the plot, the shape of the line is quite the same as the raw one. We think these methods could be used in the next step.

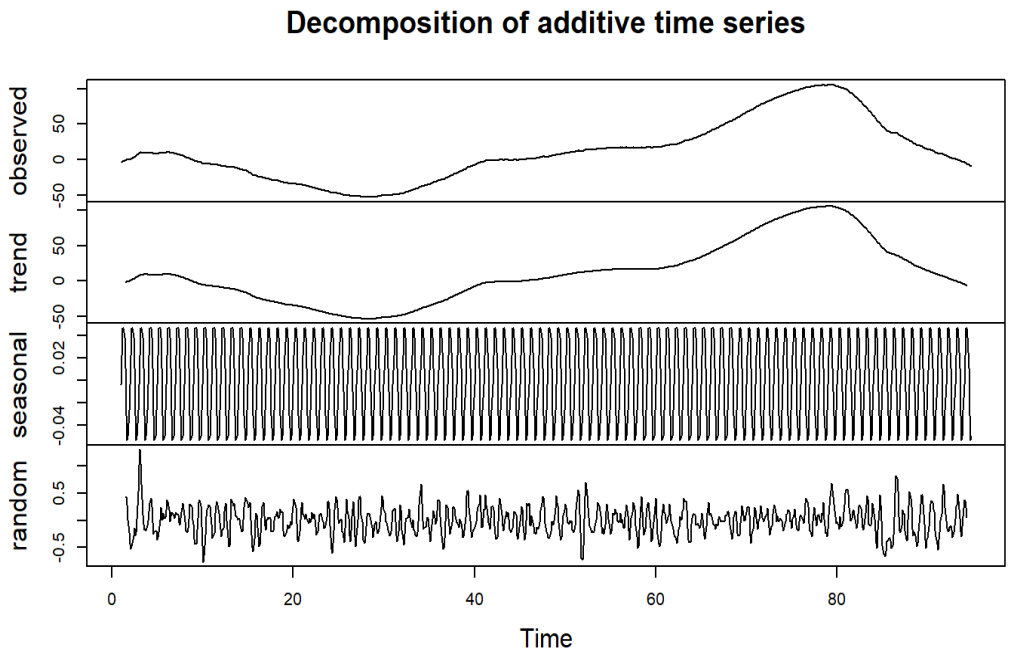


Figure 15: Decomposition of Additive Time Series

But after we dug into this method, we found it is no use in our data sets!
Because the ARIMA Model is the prediction of the future data. And the data set we have could not be predicted well as you can see the shaded part in the right plot.

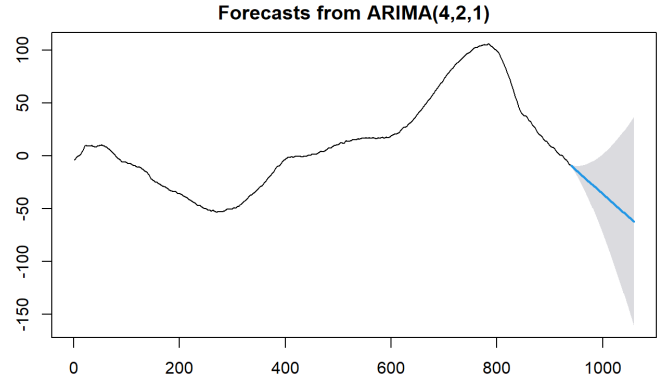
```
Series: G
ARIMA(4,2,1)

Coefficients:
         ar1     ar2      ar3      ar4      ma1
      0.5291  0.0992  -0.2912  -0.0823  -0.6971
s.e.  0.0687  0.0357   0.0358   0.0446   0.0625

sigma^2 = 0.0246:  log likelihood = 408.42
AIC=-804.83   AICc=-804.74   BIC=-775.77

        Box-Ljung test

data:  mymodel$resid
X-squared = 1.3084, df = 5, p-value = 0.9341
```



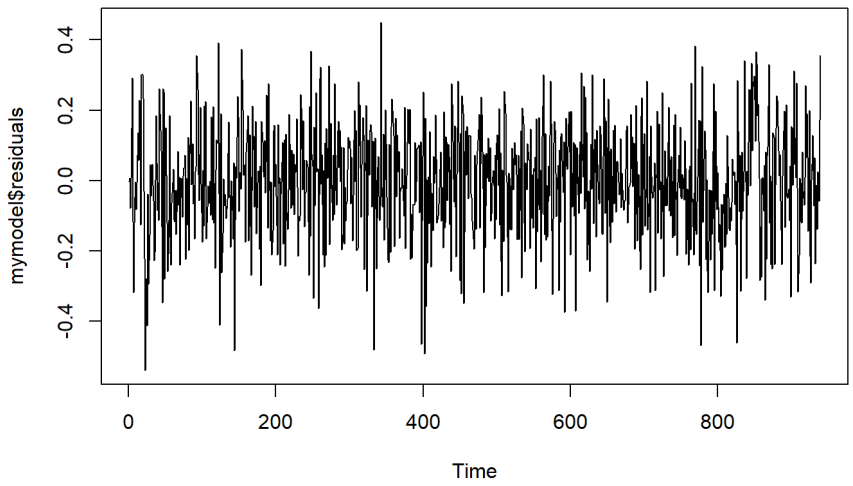


Figure 16: Analysis of ARIMA Model

According to this method, we found seasonal adjusting might be used in our project.
We have a seasonal time series that can be described using an additive model, so we can seasonally adjust the time series by estimating the seasonal component, and subtracting the estimated seasonal component from the original time series.

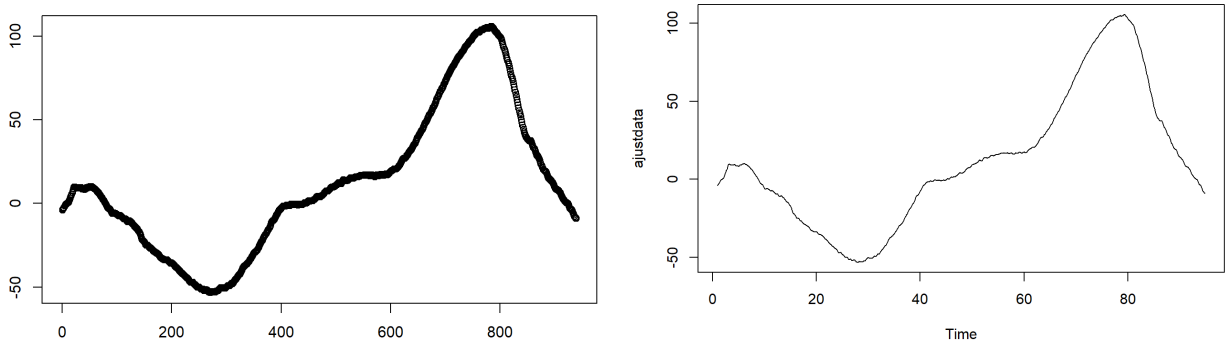


Figure 17: Real Data v.s. Seasonally Adjusting Data

Classifying Time Series

We test different kinds of neural networks (convolutional-1D and LSTM) to distinguish samples, which are generated from time series models with different knees given by ID_info.
Firstly, we try a convolutional (conv-1D) neural network with max_pooling layer, flatten layer . And we add layer_dense(units = 10, activation = 'relu') to make our network deeper. LSTM (as a kind of RNN) gives even worse accuracy.

```
 Layer (type)                                 Output Shape
Param #
=================================================================

 conv1d (Conv1D)                              (None, 231, 5)
55

 max_pooling1d (MaxPooling1D)                 (None, 57, 5)
0

 flatten (Flatten)                            (None, 285)
0

 dense_5 (Dense)                              (None, 10)
2860

 dense_4 (Dense)                              (None, 2)
22

=================================================================

Total params: 2,937
Trainable params: 2,937
Non-trainable params: 0
```

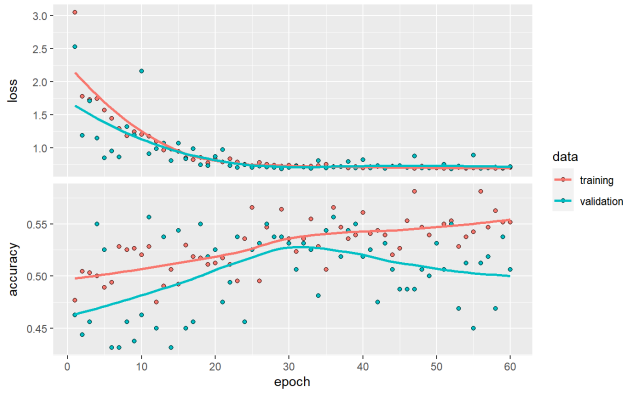


Figure 18: Conv-1D Neural Network with Accuracy = 0.52

The accuracy of these two kinds of classification are really low which is not much better than a blind guessing. The structure of the model may be adjusted in the next few days to improve the accuracy. By learning the classification of the left and right knees, we might be able to see if there is a possibility that those misclassified are patients.

```
_________________________________________________________________
 Layer (type)                                 Output Shape
Param #
=================================================================

 lstm (LSTM)                                  (None, 24)
2496

 dense_18 (Dense)                             (None, 2)
50

=================================================================

Total params: 2,546
Trainable params: 2,546
Non-trainable params: 0
```

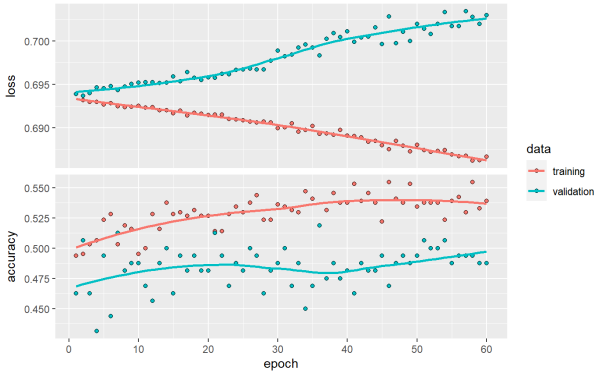


Figure 19: LSTM Model with Accuracy = 0.51

Anomaly Example Output

```
Number of anomaly samples:  247
Indices of anomaly samples:  (array([  616,   797,   797,   797,   799,   799,   799,   799,   799,
         799,   800,   856,   856,   856,   858,   858,   858,   858,
         858,   858,   859,   859,   860,   860,   860,   860,   860,
         861,   862,   862,   862,   862,   862,   862,   862,   862,
        1036,  1037,  1429,  1500,  1500,  1500,  1500,  1758,  1803,
        1804,  1804,  1805,  1806,  2245,  2245,  2245,  2245,  2245,
        2514,  2514,  2520,  2521,  2521,  3783,  3783,  3783,  3783,
        3783,  3783,  3789,  3789,  3789,  3789,  3789,  3789,  3789,
        3789,  3790,  3790,  3790,  3790,  3790,  3790,  3790,  3791,
        3791,  3791,  3791,  3792,  3792,  3792,  3792,  3792,  3792,
        3792,  3792,  3838,  3838,  3838,  3838,  3838,  3838,  3838,
        3839,  3839,  3839,  3839,  3839,  3839,  3840,  3840,  3840,
        3840,  3841,  3841,  3841,  3841,  3841,  3841,  3842,  3842,
        3843,  3843,  3843,  3843,  3843,  3996,  3997,  4059,  4730,
        4932,  4939,  4979,  4980,  4982,  4982,  4982,  4982,  4982,
        4983,  4984,  4985,  6091,  6156,  6156,  6156,  6156,  6156,
        6382,  6382,  6382,  6383,  6383,  6383,  6383,  6383,  6384,
        6384,  6384,  6384,  6384,  6384,  6384,  6384,  6385,  6385,
        6385,  6385,  6386,  6386,  6387,  6387,  6387,  6571,  6642,
        6644,  6644,  6658,  6802,  6978,  7345,  7345,  7499,  7695,
        7750,  7756,  7832,  7833,  7850,  8010,  8013,  8013,  8013,
        8013,  8013,  8385,  8385,  8385,  8385,  8385,  8464,  8466,
        8529,  8568,  8568,  8979,  9539,  9540,  9557,  9558,  9559,
        9560,  9560,  9561,  9561,  9561,  9695,  9852,  9852,  9852,
        9852,  9853,  9853,  9853,  9854,  9854,  9854,  9854,  9855,
        9975,  9975,  9975, 10648, 10648, 10648, 10648, 10648, 10648,
       11216, 12301, 12508, 12509, 12510, 12510, 12510, 12514, 12988,
       13673, 13676, 14879, 14910]), array([7, 1, 6, 7, 1, 3, 4, 5, 6, 7, 6, 5, 6, 7, 0, 1, 2, 5, 6
```

Figure 20: Example of Anomaly detection. Samples that appear multiple times have high error for multiple variables. Simultaneous samples appearing likely link back to the same subject.