Human Activity Recognition with Signal Processing

June 23, 2019

Abstract

Smartphones can capture large amounts of data concerning a user's physical movements. The University of California, Irvine posted a dataset where a smartphone was attached to participants waists and an embedded accelerometer and gyroscope was used to capture 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. A Butterworth filter was then applied to separate body acceleration and gravity. The goal is of this presentation is to use this data, to predict the transition between physical actions; this means predicting whether a person is walking, running, sitting down etc. This research implements Random Projections for dimensionality reduction as well as Fast Fourier Transforms, Cross Correlation Functions, and K-Nearest Neighbors for action class change point detection, in order to predict human activity based on smart phone triaxial sensors. This presentation will discuss the computed predicted activity as well as the computational limitations faced using JLT on a data frame of 541 features.

1 Introduction

1.1 Background

Smartphones can capture large amounts of data about a user's physical movements. Currently, smartphones can use this data to classify a user's movements into basic activities such as walking, running, or biking. These are calculated mostly through acceleration, velocity and displacement of the user. However, with the breadth of data that is available, one should be able to classify a user's actions into more precise movements such as walking upstairs or laying down. In this project, we will use signal processing to identify specific physical actions based on the features tracked by a smart phone.

1.2 Data Set

We are using a dataset from University of California, Irvine [AGO+13]. The experiments were carried out by 30 people from the ages of 19-48. Their tasks were to walk, walk upstairs/ downstairs, sit, stand and lay down while wearing a smart phone (in particular the Samsung Galaxy SII) on their waist. The phone contains an embedded accelerometer and gyroscope that is used to capture 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The data was labeled manually but partitioned randomly so that 70 % of the volunteers were selected for generating the training data and 30 % for the test data.

The sensor signals provided by the embedded accelerometer and gyroscope were pre-processed. This was done by applying noise filters and sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, were separated using a Butterworth filter (low-pass) into body acceleration and gravity. The gravitational force is assumed to have only low frequency components so a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

1.3 Goal

Our goal is to use the training data provided to:

1. Improve computational time through dimensionality reduction with random projections.
2. Implement change point detection to determine when the action class changes.
3. Determine the action class between each class change using K-nearest neighbours.
4. Evaluate error at each step.

Other research has been done on this data [Boa18], but the signal processing approach, after using random projections with change point detection, has not been done before. Existing research on this data includes implementing Naive Bayes Classifiers, Neural Nets, and Support Vector Machines, all of which use background knowledge about the features in the data to reduce the dimensions manually and assume that the intervals between activities are already known.

2 Methodology

For our project we wrote an R script [BR18] for all computations to manipulate our data. We made use of several existing packages which are referenced both in this paper and in the source code.

2.1 Random Projections

Lemma 2.1. *(Johnson-Lindenstrauss). Let* $\mathbf{x_1}, ..., \mathbf{x_m} \in \mathbb{R}^n, m \geq 2,$ *and* $0 < \delta, \epsilon < 1$.
If

$$k < C \cdot \delta^{-2} \cdot \log(m/\epsilon),$$

for some universal constant C>0, then there exists a matrix $\mathbf{W} \in \mathbb{R}^{n \times k}$ *such that*

$$(1-\delta)||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2 \leq ||\boldsymbol{W}^\top(\boldsymbol{x}_i - \boldsymbol{x}_j)||_2^2 \leq (1+\delta)||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2, \forall i, j = 1, ..., k$$

with probability at least $1 - \epsilon$. *Specifically,* $\mathbf{W}^\top$ *can be taken as* $\mathbf{W}^\top = \frac{1}{\sqrt{k}}\mathbf{A}$, *where* $\mathbf{A} \in \mathbb{R}^{\mathbf{k} \times \mathbf{n}}$ *is a Gaussian random matrix. [Adc18]*

Random projections are a dimensionality reduction technique that uses the Johnson–Lindenstrauss Theorem to reconstruct features onto random dimensions within an error constraint. Like other dimensionality reduction tools, random projections work best on data whose original features have some correlation, indicating that a smaller subset of features will be able to describe the existing relationships.
Within random projections, there are several methods. Some common choices are the Gaussian, Probability, Achlioptas, and Li methods. They are as follows:

1. "**Gaussian**" - Gaussian distribution also known as normal distribution is the default projection function. It is a continuous probability distribution used to represent real-valued random variables. The elements in the random matrix are drawn from $N(0, \frac{1}{k})$, where $N \in \mathbb{N}$ and the k value is calculated based on Johnson-Lindenstrauss Lemma.

2. "**Probability**" - For this random projection method, the matrix was generated using the uniform probability distribution with the elements inside the interval $[-1, 1]$.

3. "**Achlioptas**" - This random projections technique gives a matrix called the Achlioptas matrix. This matrix is easy to generate, has the property s= $\frac{1}{3}$ sparse, thus we cut-off $\frac{2}{3}$'s of the amount of computation. [Ach01]

4. "**Li**" - This is a generalization of the Achlioptas method that is used to generate very sparse random matrix. This improves the computation speed of random projections. [LHC06]

We decided against the Gaussian and Probability random projections due to the fact that the two methods do not produce sparse matrices, which are desired for improved computation time. Between the Achlioptas and Li methods, we chose Achlioptas because it has better documentation of past implementations in most programming languages.
We determined that the minimum number of dimensions required by JLT for a 50% epsilon is 132.

$$\begin{aligned} d &= \lceil 4\frac{\log(n)}{\epsilon^2} \rceil \\ &= \lceil 4\frac{\log(561)}{0.5^2} \rceil \\ &= 132 \end{aligned}$$

2.2 Change Point Detection

Change points are the points of a time series where abrupt variations occur within the time series data. Such changes may represent transitions that occur between states, in our case the transition between walking, running, laying down, etc.

Many change point algorithms currently exist and have been implemented [SA17]. We implemented the divisive hierarchical estimation algorithm for multivariate analysis, which was implemented in R by Wenyu Zhang in the package ecp [NAJ14].

The divisive implementation (as opposed to the agglomerative implementation) starts with the whole time period as one state and then continually subdivides at an optimal point that maximizes the difference in mean and variance in the time series on either side until any further subdivision would not be statistically significant based on a provided significance level, alpha.

While it is common practice in many fields to choose an alpha of 0.05 due to the usual case that a false positive is worse than a false negative, in our case such a significance level does not catch enough change points. We chose to use a significance level of 0.5 such that we detect a change point when the probability of observing the change in mean and variance that occurred given that the state remains constant is less than 50% .

2.3 Fourier Transform

The idea of Fourier analysis is to break a function into a combination of simpler functions, which correspond to our frequencies. The Fourier transform or Discrete Fourier Transform, is used because we are computing using R. María Pereyra [MCP12] provides an excellent description of Discrete Fourier Transform by the following:

Fix a positive integer N, and let $\mathbb{Z}_{\mathbb{N}} := \{0, 1, ..., N-1\}$. The building blocks of the Discrete Fourier Transform are the N discrete trigonometric functions $e_n : \mathbb{Z}_{\mathbb{N}} \to \mathbb{C}$ given by

$$e_n(k) := \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} z_k \omega^{-kn} \text{ for } n \in \mathbb{Z}_{\mathbb{N}}$$

The discrete Fourier Transform of a given vector $v = [z_0, ..., z_{N-1}] \in \mathbb{C}^N$ is calculated by applying a certain invertible NXN matrix to v, and the Discrete Inverse Fourier Transform by applying the inverse matrix to the transformed vector. This invertible matrix is defined below:

Definition 2.1. The Fourier matrix F_N is the matrix with entries :

$$F_N(m, n) = e^{2\pi imn/N} \text{ for } m, n \in \{0, 1, ..., N-1\}$$

The discrete Fourier transform of v is given by the following:

$$\hat{v} := \frac{1}{\sqrt{N}} \overline{F_N} v.$$

The matrix $\frac{1}{\sqrt{N}} \overline{F_N}$ is unitary, because its columns are orthonormal vectors. Therefore the Discrete Inverse Fourier transform can be written as :

$$v := \frac{1}{\sqrt{N}} F_N \hat{v}.$$

The Fast Fourier Transform is based on a factorization of the Fourier matrix into a product of sparse matrices. The sparsity of the matrix breaks the computation time down. We can compare the methods as follows:

- the Discrete Fourier Transform (DFT) which requires $O(N^2)$ operations (for n samples)
- the Fast Fourier Transform (FFT) which requires $O(N \cdot \log_2(N))$ operations

Where N is the length of the signal. We chose to use the FFT method, due to the efficiency in computation time, via an R package created by João Neto [Net13]

2.4 Spectral Coherence

The method that we first attempted to compare the closeness of two Fourier time series is called Spectral Coherence, or Cross Spectral Density (CSD). We define the Cross Spectral Density as the Fourier transform of the cross-covariance[Pen00] :

$$\mathbb{P}_{i,j} = \sum_{n=-\infty}^{\infty} \sigma_{x_i x_j}(n) \exp(-iwn)$$

where x_i and x_j are time series functions and the cross-covariance of a signal is given by :

$$\sigma_{x_i x_j}(n) = \sum_{n=-\infty}^{\infty} x_i(l) x_j(l-n)$$

This gives us:

$$\mathbb{P}_{i,j} = \sum_{n=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x_i(l) x_j(l-n) \exp(-iwn).$$

Let $k = l-n$, we can see that $\exp(-iwn) = \exp(-iwl)\exp(-iwk)$. Thus giving us a new expression for CSD which is as follows:

$$\mathbb{P}_{i,j} = X_i(w) X_j(-w)$$

where

$$X_i(w) = \sum_{l=-\infty}^{\infty} x_i(l) \exp(-iwl)$$

$$X_j(w) = \sum_{k=-\infty}^{\infty} x_j(k) \exp(+iwk)$$

Note that for real signals we have that $\overline{X_j(w)} = X_j(-w)$. Hence we can rewrite it once again as follows:

$$\mathbb{P}_{i,j} = X_i(w)\overline{X_j(w)}.$$

This means that the CSD can be evaluated in one of two ways:

1. by first estimating the cross-covariance and Fourier transforming or
2. by taking the Fourier transforms of each signal and multiplying (after taking the conjugate of one of them).

Once we computed the FFT of our data, we had an array with dimensions $299 \times 3 \times 128$, where the 299 rows are the unique unidentified activities, the 3 columns are the random projection features, and the third dimension of length 128 are the Inverse Fast Fourier Transform results. We computed the spectral coherence [Sha18] between two activities by taking the spectral coherence between each possible pair of the three random projection features in each time series and averaging them. To account for the potential of unaligned lags, the spectral coherence should be iteratively computed for each possible lag and the maximum of these Spectral Coherences was used as the final results.

While we could consider each activity now as a 3×128 matrix and compute the closeness of each activity using a matrix norm to set up our problem for supervised learning, we don't know that each activity is the same length, and any oscillations detected may have different amounts of lag compared to other oscillations for the same activity. This renders the matrix norm as a measure of closeness between activities useless.

At this point in our analysis, we were no longer able to proceed with spectral coherence because the implementation of the lag optimization was beyond the scope of this project.

2.5 Cross Correlation Functions

After finding some major hurdles in the spectral coherence method, we used the another technique to measure the distance: the Cross Correlation function. The Cross Correlation tells us whether there is a strong correlation between two time series for a number of different lag options. It is defined as follows [Wei]:
The cross correlation of two complex functions $f(t)$ and $g(t), t \in \mathbb{R}$ denoted $f \star g$ is:

$$f \star g = \overline{f}(-t) * g(t)$$

where $\overline{f}(t)$ is the complex conjugate of $f(t)$ and $*$ denotes the convolution, which is defined as:

$$f * g = \int_{-\infty}^{\infty} f(x)g(t-x)dx$$

Thus it follows that

$$f \star g = \int_{-\infty}^{\infty} \overline{f}(x)g(t+x)dx.$$

R has an implementation of this function, which we used. Running this function allowed us to determine the lag at which the correlation between two time series is strongest. We took the supremum of the cross correlation function for different inputs of lag as our measure of closeness.

2.6 k-Nearest-Neighbors

Finally, after processing our data, we used k-Nearest Neighbors for classification. In pattern recognition, the k-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k-Nearest-Neighbors. If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

The following is the pseudo code [THO14] for k-NN:

```
let : $\mathbf{X}$ represent the training data
$\mathbf{Y}$ represent the class labels of $\mathbf{X}$ and
$x$ be an unknown sample.
Classify $(\mathbf{X}, \mathbf{Y}, x)$
for i= 1 to m do :
    compute distance d($\mathbf{X}_i$ ,x)
end for
Compute set $I$ containing indices for the k smallest distances $d(\mathbf{X}_i, x)$
return majority label for $\{\mathbf{Y}_i$ where $i \in I\}$
```

We adapted the K-Nearest Neighbors algorithm to use the cross correlation function as the distance function. This results in a matrix which contains the distances between each of the predicted activity time series and each of the actual activity time series. Using k=5, we find the 5 closest labeled time series based on the cross correlation function for each of the predicted time series. We used the most common event among the 5 closest neighbors as our prediction. We completed this final step with the help of the built-in function k.nearest.neighbors() from the FastKNN library in R to find the set of k indices that contain the smallest distances.

3 Results

3.1 Random Projections Results

Figure 1 shows that strong negative and positive correlations exist in our original data (seen by the dark blue and deep red squares in the correlation plot). We only plotted the first 100 features for brevity.

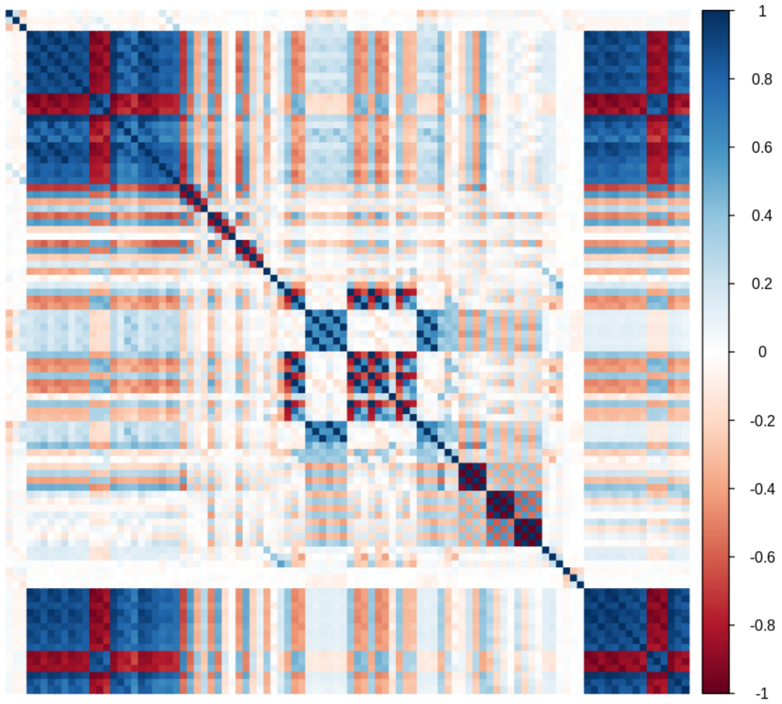


Figure 1: Correlation plot of the first 100 features

3.2 Change Point Detection Results

Figure 2 shows the actual change points in blue and the predicted change points in red. While only some of the actual change points are detected, those that are fall fairly close to the predicted change points.

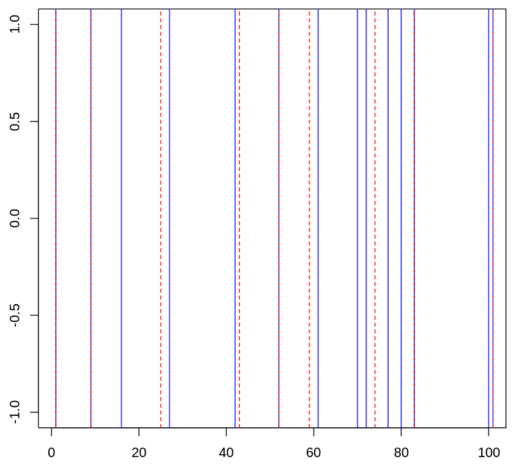


Figure 2: Actual (Blue) and Predicted (Red) Change Points for the First 100 Observations

Our method does not capture every change point, so this will increase the error in our human activity recognition predictions going forward. The sum of squares error is 122.7721 and the mean squared error is 0.3077.

The following is the method used to find the error of the actual change points as compared to the predicted change points.

Let C_i be the i^{th} change point and $\hat{C_i}$ be the predicted change point closest to C_i.

$$SSE = \sum_{i=1}^{n} (C_i - \hat{C_i})^2 = 122.7721$$

$$MSE = \frac{1}{n}(SSE) = \frac{1}{n}\sum_{i=1}^{n} (C_i - \hat{C_i})^2 = 0.3077$$

Figure 3 shows that most of the errors are less than one time point in length. However, the distribution of the errors has a strong right skew and these higher skewed errors have a considerable effect on the mean squared error. The skew comes from the nature of the change point detection; when the change point is detected it is usually predicted exactly or very close to the actual point However, when the change point is missed, then the closest predicted change point won't occur until the Human has changed to the next activity, which may be a long period of time.

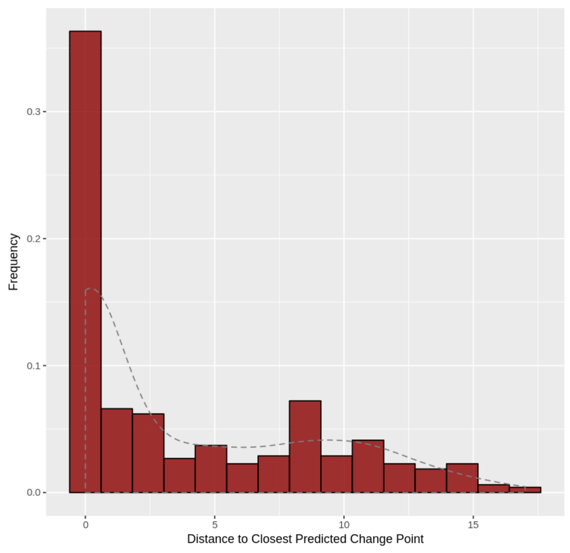
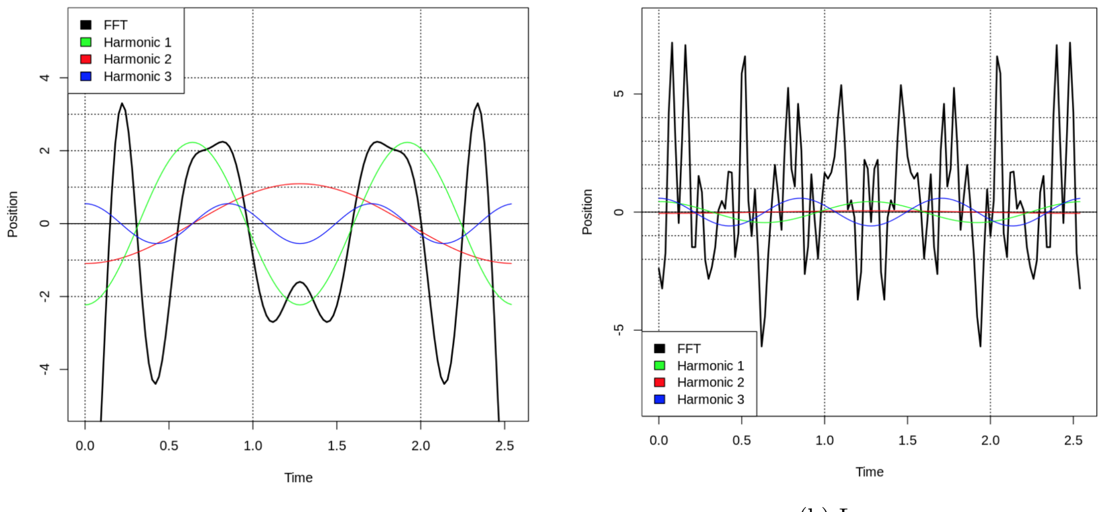


Figure 3: Histogram of the Observed Errors for Change Point Detection

3.3 FFT Results

Once we obtained the predicted change points, we applied an inverse Fast Fourier Transform to each time series between change points [Net13] to obtain a function for the time series. This enables us to compare functions of time series of different lengths. The Inverse Fast Fourier Transforms and their 1st (red), 2nd (green), and 3rd (blue) harmonic oscillations for a short unidentified activity with 8 recordings and a long unidentified activity with 41 recordings are shown in Figure 4. Figure 4 only shows the Fourier transform for the first random projection feature as an illustration; all features are used to classify the event.



(a) Short (b) Long

Figure 4: Short and Long Fourier Series for Unidentified Activity Time Series

Based on figure 5 we see that our reconstruction of the signal has a lot of noise. The frequency domain shows the voltages present at varying frequencies; a different way to look at the same signal. However, when you view the signal in the frequency domain, you expect only one spike because you are expecting to output a single sine wave at only one frequency. The frequency domain is great at showing if a clean signal in the time domain actually contains cross talk, noise, or jitter.

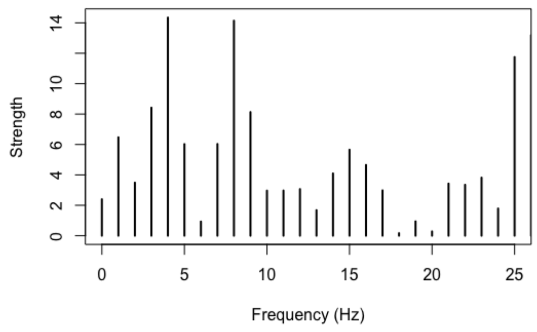


Figure 5: Frequency Spectrum plot of wave

An inverse Fourier Transform (IFT) converts the frequency domain components back into the original time wave. We use the Inverse Fast Fourier transform process to create two datasets: one for the unidentified activities based on detected change points and the other for the known activities based on actual change points.

3.4 Cross Correlation Function Results

Figure 6 shows that the maximum cross correlation function between known activity 1 and unknown activity 1 exists at lag 0. We use the absolute value maximum cross correlation function across lags as the measure of closeness between two time series, averaged for the pairwise comparisons between the three random projections of each time series.

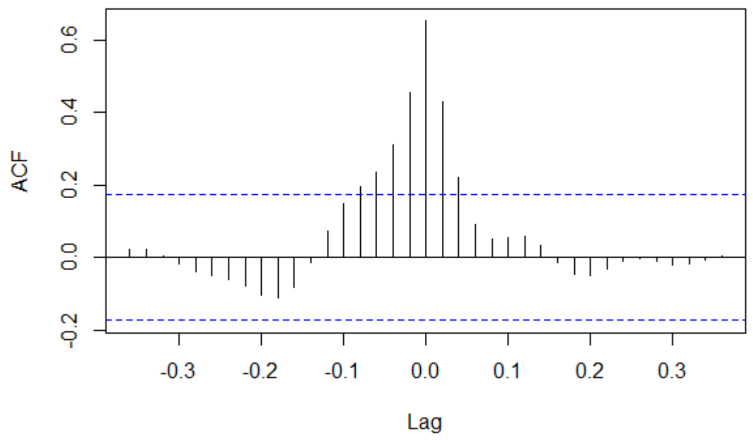


Figure 6: Unknown Activity 1 and Known Activity 1 CCF

3.5 K-Nearest-Neighbors Results

Our model has an overall accuracy of 19.48% of predicting the correct event for every measurement in time.

Figure 7 shows the predicted and actual activity class labels. Most activities were predicted as either walking or laying, despite being a wide variety of actual activities. Our model almost never predicts sitting. This may be because the stillness associated with sitting is very difficult to distinguish between the stillness associated with laying down.

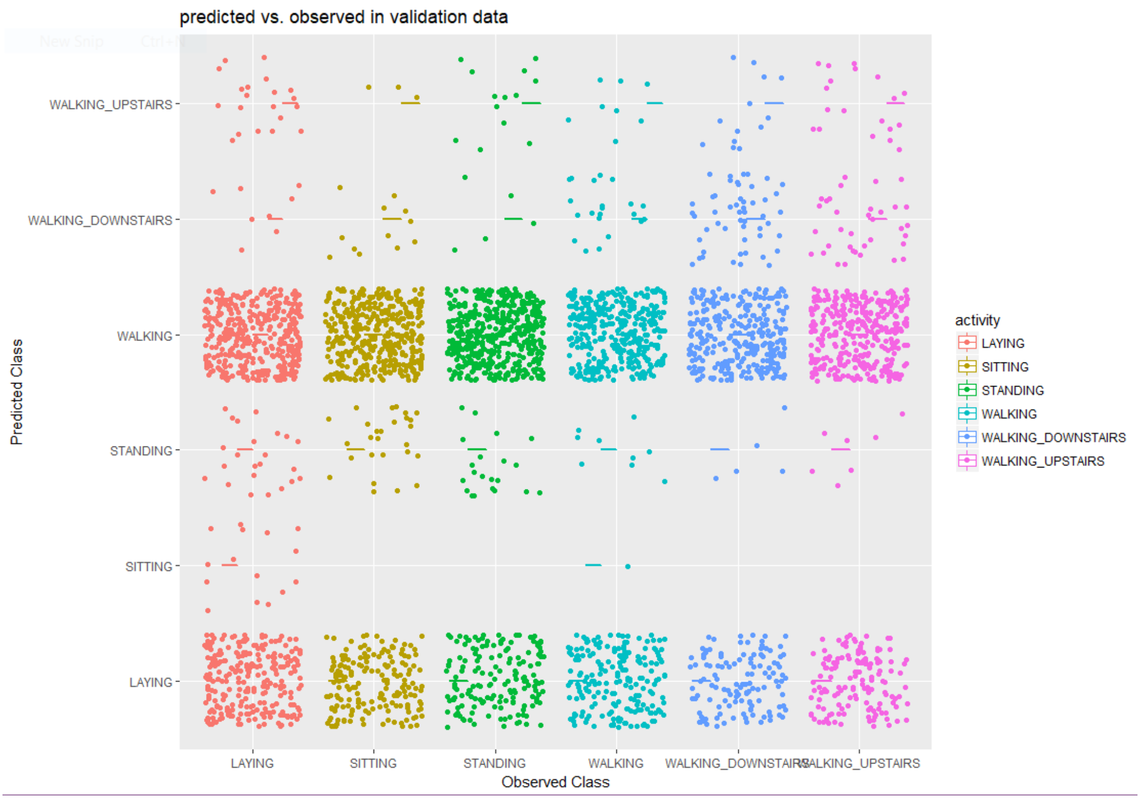


Figure 7: Confusion Plot of Classification Results

4 Discussion

While we wanted to implement Random Projections on our data with the minimum numbers of dimensions $d = 132$ as outlined by JLT, practical limitations of our available computing resources and the time constraints of this project meant that we were unable to implement further algorithms with such a large number of dimensions. After some reflection, we decided to continue the analysis using random projections onto an $\mathbb{R}^3$ space, realizing that this comes at a large loss in the variance we are able to represent in our data. Our hope is that at some point in the future we can replicate this work with the 132 dimensions as recommended and find a much higher accuracy rate. This could come down to optimizing the code for efficiency, or submitting the job to a spark cluster.

While the long computation time required for this number of dimensions may seem undesirable, since our long term goal is for this methodology to be usable in an online situation, each new activity will take much less computation time to match to its k nearest neighbors using 132 dimensions. The problem with our code is that we have to do this for 229 unlabeled activities to get a sense of our overall model accuracy is extremely costly.

One of the major problems with using random projections is that we lose the meaning behind the signals of the data and the ability to model the physical relationships using mechanics. However, exploring this problem with random projections allows for us to model the relationships with without background knowledge about the mechanics.

While the dimensionality reduction could have also been carried out through means of principle component analysis, we preferred random projections because PCA becomes expensive with a large number of features (562). Our application for signal processing of human activities must be able to make predictions on new data in real time so computational efficiency becomes imperative. However in hindsight, knowing that we would not be able to proceed with the minimum number of dimensions by JLT, PCA may have produced much more accurate results.

We chose the divisive change point detection algorithm because of our need for an algorithm that accommodates multivariate time series. Although using multivariate data is fairly computationally expensive, none of the features in the original data can accurately predict the change points on their own so it is necessary. If we were to use the full 132 random projection dimensions, this is the step that would have become especially computationally expensive. There may be a possible implementation of change point detection for multivariate data that is more efficient, however designing such an implementation was beyond the scope of this paper.

References

[Ach01] Dimitris Achlioptas. Database-friendly random projections, http://doi.acm.org/10.1145/375551.375608, 2001.

[Adc18] Ben Adcock. Math 496- lecture notes 5 random projections, 2018.

[AGO$^+$13] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones, https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones, 2013.

[Boa18] Marcos Boaglio. Simplified human activity recognition w/smartphone, https://www.kaggle.com/mboaglio/simplifiedhuarus/kernels, 2018.

[BR18] Kristen Bystrom and Alice Roberts. Human activity recognition, https://github.com/ksbystrom/human-activity-recognition, 2018.

[LHC06] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections, 2006.

[MCP12] Lesley A. Ward Maria Cristina Pereyra. Harmonic analysis from fourier to wavelets, 2012.

[NAJ14] David S. Matteson Nicholas A. James. ecp: An r package for non-parametric multiple change point analysis of multivariate data, https://cran.r-project.org/web/packages/ecp/vignettes/ecp.pdf, 2014.

[Net13] João Neto. Fourier transform: A r tutorial, http://www.di.fc.ul.pt/ jpn/r/fourier/-fourier.html, 2013.

[Pen00] William D. Penny. Signal processing course, https://www.fil.ion.ucl.ac.uk/ wpenny/-course/course.html, 2000.

[SA17] Diane J. Cook Samaneh Aminikhanghahi. A survey of methods for time series change point detection, https://www.ncbi.nlm.nih.gov/pmc/articles/pmc5464762/, 2017.

[Sha18] Gillian Sharer. crossspectrum, https://www.rdocumentation.org/packages/ irisseis-mic/versions/1.4.8/topics/crossspectrum, 2018.

[THO14] Bunheang Tay, Jung Keun Hyun, and Sejong Oh. A machine learning approach for specification of spinal cord injuries using fractional anisotropy values obtained from diffusion tensor images, 01 2014.

[Wei] Eric W. Weisstein. Cross-correlation from mathworld–a wolfram web resource. http://mathworld.wolfram.com/cross-correlation.html.

5 Appendix

5.1 Achlioptas Random Projection Algorithm

```
# Random projections to a lower dimension subspace with the Achlioptas' projection matrix
# The projection is performed using a projection matrix P s.t.
    Prob(P[i,j]=sqrt(3))=Prob(P[i,j]=-sqrt(3)=1 6;
# Prob(P[i,j]=0)=2 3
# Input:
# d : subspace dimension
# m : data matrix (rows are features and columns are examples)
# scaling : if TRUE (default) scaling is performed
# Output:
# data matrix (dimension d X ncol(m)) of the examples projected in a d-dimensional random
    subspace
Achlioptas.random.projection            (d=2, m, scaling=TRUE){
  d.original        (m)
     (d >= d.original)
        ("norm.random.projection: subspace dimension must be lower than space dimension",
            .=FALSE)
  # Projection matrix
  P          (     (d d.original,1,7)); # generate a vector 1 to 6 valued
  sqr3        (3)
  P[P==1]    sqr3
  P[P==6]    -sqr3
  P[P==2 | P==3 | P==4 | P==5]    0
  P           (P,     =d)
  # random data projection
     (scaling == TRUE)
    reduced.m        (1 d)   (P %   m)

    reduced.m    P %   m
  reduced.m
  }
```