Time Series Classification Through Shapelet Transformations

Luke Davidson
M.S. in Robotics Student at Northeastern University
davidson.lu@northeastern.edu

Abstract

The classification of time series data is an area of machine learning that has proved to be extremely beneficial and fit for everyday use throughout various industries in recent years. It is a distinct field of machine learning that is often approached using non-standard methods due to the uniqueness and variability of the input. In this study, I explore the method of creating and comparing shapelet transformations to classify specific physical actions performed by a group of human test subjects. I aim to create an improved method for shapelet learning from labeled time series data and explore the difference in performance between this improved model and generic shapelet comparison methods. I show that through the use of improved models, such as averaging shapelet transformations, increases in accuracy can be achieved. I also show the limitations to classification via shapelet transformations as it relates to physical motion of the human body, and how these limitations can be best minimized.

Objectives and Significance

The overall goal of time series classification methods is to detect patterns in individual motions that are representative of certain classes and compare those patterns to new unlabeled series data to help predict and better the outcome of an observed event [1]. The extent of time series data in everyday life is often overlooked. Whether it is through analyzing the heartbeat of a patient, monitoring speeds and locations of an autonomous vehicle driving on a highway, modeling financial markets, or tracking a professional athlete's workout through a smart watch, time series data is prevalent in arguably every industry there is. Time series classification is an extremely important field of machine learning due to its wide range of applications and ability to provide meaningful, and sometimes even life changing, predictions and analytics. The early detection of an irregular heartbeat in a patient, or the ability for an autonomous car to accurately control its navigation on a highway, can ultimately save an individual's life. Other applications provide industry changing analytics through the analysis and classification of certain time series trends. Classifying stock and market trends can help predict important financial events and

inform a trader about optimal actions, and the classification and analysis of the physical motion of an athlete can lead to improved fitness and performance analytics.

The goal of this study is not only to exploit known techniques for classifying time series data, but to modify aspects of existing methods to obtain improved accuracy results and resolve limitations that are often seen in the field. Specifically, I aim to face the challenging problem of classifying human motion of specific actions, oftentimes that are very similar to each other (ex. punching vs. slapping). As the combination of an athlete and robotics student, I have always been extremely interested in the relationship between physical motion and data analytics. High quality robotic mechanisms consist of joints, links, and end effectors that are driven by motors based on sensory information obtained from equipped sensors [9]. Human limbs can be described in the same exact way: the links being bones, the end effectors being hands and feet, the motors being muscle, and the sensors being the senses, all analyzed by the brain. The performance of robotic mechanisms is often analyzed through computations such as throughput and efficiency. When sensors are attached to the human body, the physical performance of a human can be analyzed using the same computations. This study only scratches the surface of my interests involving the relationship between these two fields, but displays the possibilities of how much can be learned when applying machine learning techniques to the study of physical human motion.

In this study, I find that modifications to existing time series classification techniques prove to be more beneficial than standard methods when facing certain limitations. I also highlight those limitations, and offer explanations as to what can further be done to eliminate those constraints and improve the relationship between analytics and the study of physical human motion.

Background

One of the most popular classification algorithms used to solve time series classification problems is that of shapelet transformations. Shapelet transformation algorithms aim to search through sequences of labeled time series data and identify subsequences that are maximally representative of the respective class [1]. These subsequences are referred to as shapelets [1]. Figure 1 below shows an example of a possible shapelet, Q, compared to a series, C, of the same class.

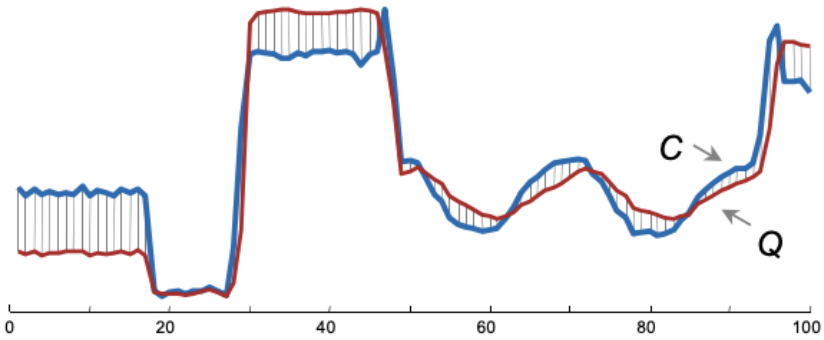

Figure 1: Series data C being compared to shapelet Q of the same class

In a typical shapelet transformation problem, a set of shapelets are identified for multiple classes across a certain domain. When new, unlabeled data is observed in the form of a series belonging to the same domain, each class representative shapelet is compared to the unlabeled data and an error is calculated between the two. In the simplest form, the unlabeled series is assigned to the class in which the shapelet comparison results in the lowest error. There are a few error calculations that can be computed between the data and the respective shapelet, the most common being a sum of squared Euclidean distance [2]. For a shapelet s, and unlabeled data series d, both of length t, the sum of squared Euclidean distance is represented by the equation:

$$ED_{ds} = \sum_{i=1}^{t}(d_i - s_i)^2$$

Eq 1: Sum of squares of Euclidean Distance

where the assigned class described by the decision criterion above would then be represented by:

$$argmin_{y \in Y}\{\sum_{i=1}^{t}(d_i - s_{yi})^2\}$$

Eq 2: Classification based SSE

There are various ways in which to determine what these representative shapelets are, how to accurately search for and compare them to unlabeled data, and ways in which to make them scale and shift invariant, all of which grasp the scope of this study and are explained in the following sections.

Shapelet transformations are one of the leading algorithms related to time series classification problems due to a few major advantages [2]. First, they can be easily applied to a wide range of problem statements. As mentioned above, time series data is prevalent in some form in arguably every industry. From healthcare, to finance, to nature, to fitness, the need to be able to identify representative patterns and

sequences in time series data is endless. Second, shapelet transformations can provide extremely interpretable results compared to other methods of classification [2]. See Figure 2 below for an example.

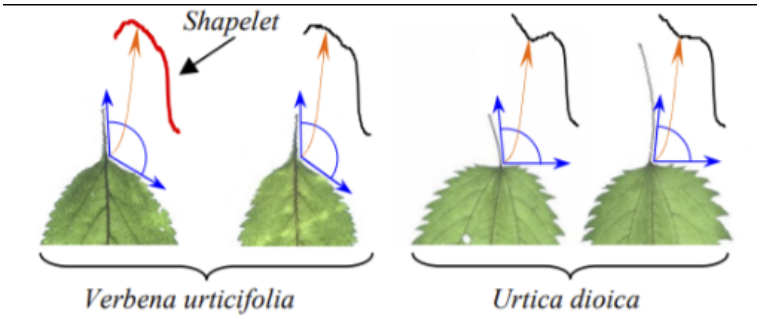

Figure 2: Shapelets of two very similar different species of leaves

In Figure 2, shapelets are used to determine the class of a given leaf. The verbena urticifolia and urtica dioica leaves appear to be very similar in appearance, with the defining factor being the angle that the stem makes with the body. In this problem statement, results can be read as "Does the leaf make a 90 degree angle with the body?" and a decision tree can be constructed with yes or no answers. This makes classification results extremely interpretable to any researcher. Finally, much work has been done in the field of shapelets related to their transformations [3]. It is trivial to simply compare shapes together and calculate an error difference. Much thought has to go into the comparison itself. Relating to the classification of human motion, what will the positional series data look like in a 7 foot human kicking an object compared to a 5 foot human kicking an object? What if the 5 foot human is much quicker than the 7 foot human? One might imagine these series plots will be different in scale and frequency, yet they should still represent the same class. This example represents the importance of scale invariance [3], a common issue that has been greatly studied in the field of shapelet transformations due to its frequent appearance.

In this study, I aim to resolve some of the aforementioned limitations using a custom approach involving the scaling, shifting, and averaging of multiple shapelet transforms to minimize error and maximize classification accuracy on the given data set. These methods are further explained in the following sections. I also aim to purposefully confront some of these common issues seen in shapelet transformations by using a difficult to classify dataset. This data set, further explained in the following section, includes positional data from physical action very similar to each other, for example front-kicking and side-kicking. Previous work on this data set primarily involves the classification of actions using different structures of neural networks and decision trees [4][5], and has had less to do with attempting shapelet transformation algorithms.

Methods

As mentioned above, in this study I aim to develop an accurate method to classify time series data of specific physical actions performed by humans through shapelet transformations. I have obtained a data set form the UCI Machine Learning Repository [6] that includes a collection of time series positional data sequences. The data set includes positional sequence data from 10 subjects: 7 male, 3 female, ranging in age from 25-30. Each subject had 9 markers placed on their body for 3D data acquisition: 2 on each limb, and 1 on the head. X, Y, and Z coordinate data was collected for each marker while each subject performed 20 individual actions: 10 "normal" actions and 10 "aggressive" actions, defined below:

Normal: {Bowing, Clapping, Handshaking, Hugging, Jumping, Running, Seating, Standing, Walking, Waving}

Aggressive: {Elbowing, Front-kicking, Hammering, Headering, Kneeing, Pulling, Punching, Pushing, Side-kicking, Slapping}

Each subject performed each action for approximately 10 seconds with data being sampled at approximately 200 Hz, leading to roughly 2000 data points for each action. Multiple visuals of the raw data can be seen below in Figures 3 and 4.

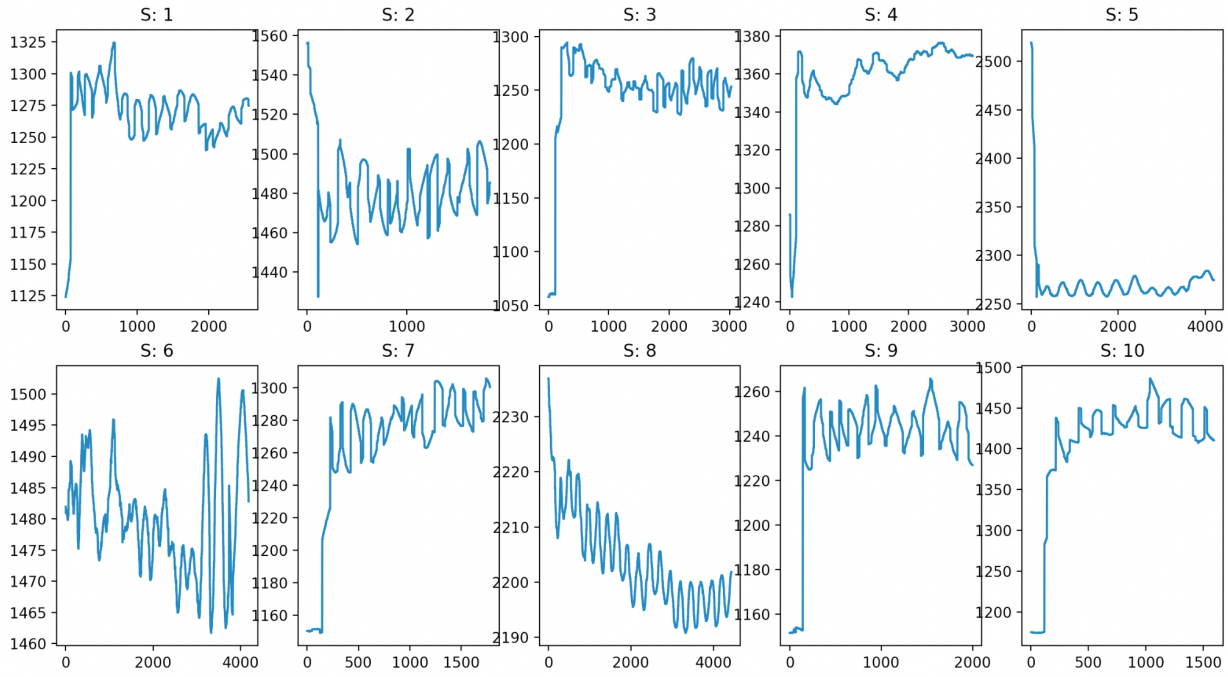

Figure 3: Raw combined positional data for all subjects performing the action "Handshaking", recorded from a marker on the right arm

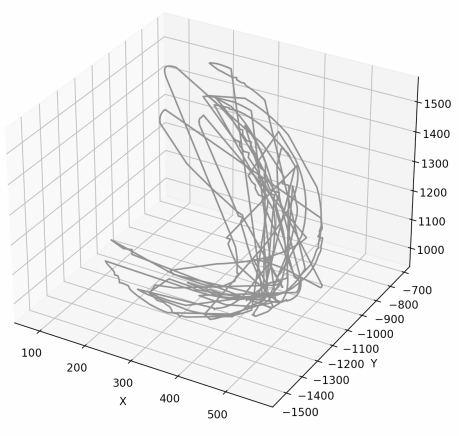

Figure 4: 3D positional data of subject 3 elbowing, data recorded from a maker on the right arm

The first phase in my implementation was the data pre-processing phase. The main steps in my pre-processing phase included:

1. Read raw data
2. Cleanse data by replace zeros
3. Combine x, y, and z coordinates
4. Normalize

The first step is evident and involves reading all of the raw data into a data frame, accompanied with columns to identify the subject, action type (normal, aggressive), action, and timestamp.

The second step is the cleansing process of replacing zeros in the data. Zeros in the data are representative of timesteps in which data was not recorded properly, leading to a default entry of 0. The effects of these false data points can clearly be seen in subjects 1, 4, 5, and 10 below in Figure 5:

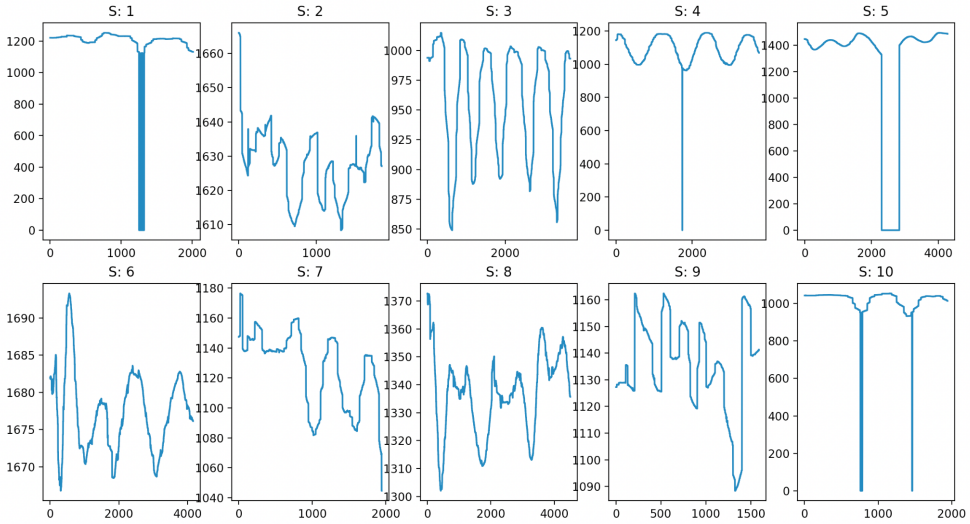

Figure 5: All subject data for Bowing, showing false data points in subjects 1, 4, 5, 10

When two shapelets are compared, say for example subject 3 and 4 in the above figure, the error measurement between each will be extremely large due to these false data points, when in reality the true aspects of the measurements match well. A pseudocode for the process of replacing zeros is shown below:

```
# REPLACING ZEROS

Y, X ← coordinates in which any data measurement is 0

for every (y, x) combo in zip(Y, X):
        y_nonzero ← y index of closest data point along x that is a true reading (non-zero)
        data[y, x] = data[y_nonzero, x]
```

This process searches for all indexes where the raw data reading is 0 (false reading), and replaces that reading with the closest true reading of that axis.

The next step in the pre-processing phase is to combine all x, y, z coordinates for each marker into a combined calculation. This calculation is done using the following transformation:

$$C = \sqrt{X^2 + Y^2 + Z^2}$$

Eq 3: C represents the combined axis

Certain actions will be more prevalent along certain axes, which is almost impossible to know without hard searching through every single axis of every single action. Creating this combined axis is a solid representation of all axes of the data and allows simple conversion to 2D time series data.

The last step in the pre-processing phase is the most important: normalization. Pseudocode for the normalization process can be seen below:

```
# NORMALIZATION

for each subject:
        for each action:
                sub_act_df ← subset of data frame for the specific subject and action readings
                for each column in subject_action:
                        min ← minimum reading in sub_act_df[column]
                        sub_act_df[col] += -min         # subtract the minimum from every point to get the minimum to 0
                        max ← max reading in sub_act_df[column]
                        sub_act_df[col] /= max          # divide whole column by the max to normalize between 0 and 1
```

This normalization process comprises two main steps. First, it shifts all data by subtracting the minimum value from all points so that the minimum reading is 0. Next, it scales the whole column by dividing by the subsequent maximum value, guaranteeing all data points lie within the range [0, 1]. The importance of this normalization process is evident when analyzing Figure 5 above. If one were to calculate the error between subjects 9 and 10 from Figure 5, at first glance, it seems like they would be a near perfect fit and the error calculation would be very small. At a second glance, one notices that the upper y-axis bound for subject 9 is 1260 and for subject 10 is almost 1500. This would clearly cause a large error calculation.

This is representative of the fact that different subjects will perform an action at different forces, or with different displacements, yet the ultimate series should still represent the same class. This is an essential aspect to shapelet transformations and is resolved by the above normalization process. Figure 6 below displays the same data as Figure 5 above, after the preprocessing phase:

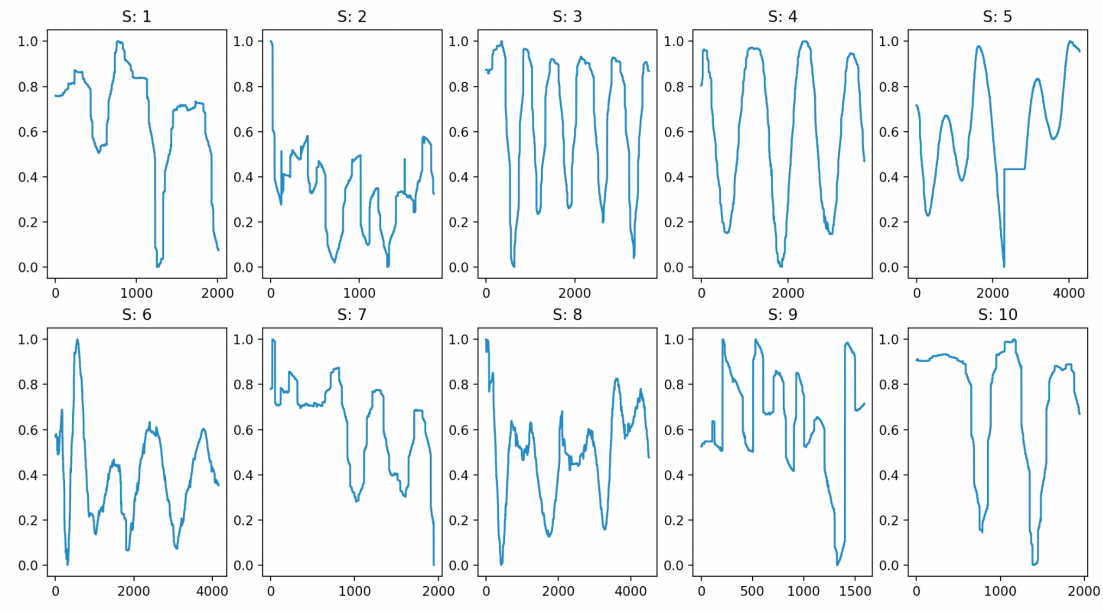

Figure 6: Visuals of same data from Figure 5, post preprocessing

Once the data is preprocessed, it is ready for the shapelet transformation and comparison phases. To limit the amount of shapelet transformations and speed up the testing process, I decided to only consider data from the limbs that corresponded best to each individual action. That is, shapelets for the action "Punching" would be created from the sensors placed on the arms, but not from the sensors placed on the legs. Similarly, shapelets for the action "Front-kicking" would be created using the sensors placed on the legs, and not the arms.

I created and tested multiple shapelet transformation algorithms, all of which I will go into more detail in the following section. The modified algorithms I implemented mainly included RMS shapelets, sub-shapelets, and averaged shapelets. The algorithms themselves have fundamental differences, although the general steps involved in the creation and comparison phases are similar. The general pseudocode for these two phases is shown below:

```
Initialize shapelets ← {}
training_data, testing_data ← split all data into training and testing data sets

# GENERAL SHAPELET CREATION
for each action/limb combo in training_data:
        shapelets[action][limb] ← create shapelet via the respective algorithm (defined in following section)
```

```
# GENERAL SHAPELET COMPARISON
for each action/limb combo in testing_data:
        sub_act_df ← subset of data frame for the specific subject and action readings in testing_data
        test_shapelet ← sub_act_df[subject][action][limb]
        for each action/limb combo in shapelets:
                target_shapelet ← shapelets[action][limb]
                for each shapelet offset:
                        error ← calculate error between test shapelet and target shapelet swept across
                        if error < min_error:
                                best_action ← action
                                min_error ← error
                accuracy ← compare best_action to true action and store results

repeat as necessary for a certain number of iterations
```

To briefly explain the above general pseudocode: all subjects are split randomly into training subjects and test subjects for each individual action + limb combo. Typically there was only one test subject, with the rest of the subjects being used as training subjects for the shapelets. A *k*-fold cross validation-like process was implemented where the whole process was repeated a number of times to ensure each subject was the test subject multiple times. Results were recorded for each individual subject for later analysis. The true "target" shapelets were then created from the training subjects data for each specific sensor using one of the algorithms further detailed below. These true targets were then swept across each test shapelet for every action+limb combination, and an error calculation was computed for each sweep. In a simplified decision criterion, the minimum error, and consequently the best action, was stored as the best fit and the test shapelet was classified as the subsequent best action.

Results

The three main shapelet transformation algorithms I implemented were the general RMS, averaged, and sub shapelet transformations. Each algorithm involves scaling and shifting different shapelets to obtain the best general fit on the training data, then sweeping that optimized shapelet over the test shapelet to identify an error, and ultimately, a classification. Each algorithm is explained in detail below, along with individual results:

RMS Shapelet Transformation

The RMS shapelet transformation is the most straightforward shapelet algorithm, yet it can be extremely effective. The RMS transformation involves the sweeping of an entire test shapelet, offset by a certain step size at each increment, over an entire training shapelet. It is called the RMS algorithm because the error calculation it computes is the Root Mean Squared Error between the overlapping sections of the shapelets. To better visualize the comparison, reference *Figure 7* below:

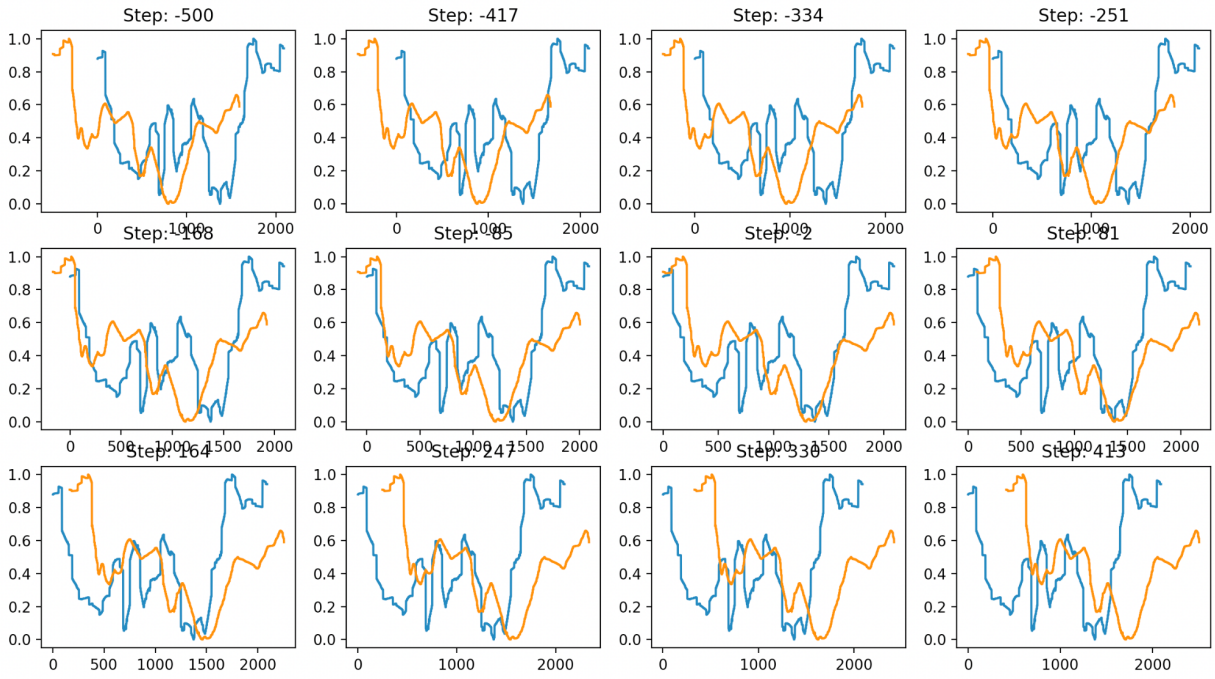

Figure 7: Target shapelet translation over a test shapelet

Figure 7 shows the translation of a learned shapelet (orange) over that of a test shapelet (blue), each originally of the same length. When the shapelets are offset by a certain step size, for example -500 in the top left plot, only a portion of each shapelet overlaps and can be used to calculate an accurate error measurement. This leads to an optimized error calculation of the RMS, given by:

$$RMS = \sqrt{\frac{\sum_{i=overlap}^{overlap-stepsize}(d_i - s_i)^2}{total - 2 \cdot stepsize}}$$

Eq 4: RMS Error for an overlapping shapelet comparison

The SSE is essentially averaged over the overlapping series timestamps for each shift. The RMS error is an optimized error calculation for this implementation of a shapelet transformation because it allows us to accurately compare datasets of different length.

The pseudocode for the RMS shapelet transformation is shown below:

```
Initialize shapelets ← {}
training_data, testing_data ← split all data into training and testing data sets

# SHAPELET COMPARISON
for each action/limb combo in testing_data:
        test_sub_act_df ← subset of data frame for the specific subject and action readings in testing_data
        test_shapelet ← test_sub_act_df[subject][action][limb]
        for each action/limb combo in training_data:
                target_sub_act_df ← subset of data frame for the specific subject and action readings in training_data
```

```
        target_shapelet ← shapelets[action][limb]
        for each shapelet offset:
            error ← calculate error between test shapelet and target shapelet swept across
            if error < min_error:
                min_error ← error
        errors ← store min_error for each training shapelet
    sorted_errors ← sort errors by minimum error
    prediction ← K-NN based prediction for each test subject
```

This algorithm is slightly different from the following algorithms as each test shapelet is compared directly to its training shapelets of the same limb. A best RMS error is stored for each comparison, along with the respective target action for that RMS error. This results in a large 2-dimensional array with RMS error in the first column and an integer representing the action number in the second column. Results for a run for action/limb combination of "Clapping" and "l_arm_m3" is shown below in Table 1:

{'Clapping_l_arm_m3': array([[5.48914750e-03, 0.00000000e+00],
[1.04344056e-02, 0.00000000e+00],
[7.46847712e-03, 0.00000000e+00],
[1.09008684e-02, 0.00000000e+00],
[6.36762422e-03, 0.00000000e+00],
[3.78948309e-03, 0.00000000e+00],
[2.74365956e-03, 1.00000000e+00],
[3.30659380e-03, 1.00000000e+00],
[3.65277945e-03, 1.00000000e+00],
[3.20706991e-03, 1.00000000e+00],
[4.10392749e-03, 1.00000000e+00],
[2.95625025e-03, 1.00000000e+00],
[4.03873599e-03, 1.00000000e+00],
[8.89692405e-03, 2.00000000e+00],
[1.30029617e-02, 2.00000000e+00],
……
[8.66956781e-03, 1.10000000e+01],
[1.04765635e-02, 1.10000000e+01],
[9.59627259e-03, 1.10000000e+01],
[7.88579730e-03, 1.20000000e+01],
[8.11045152e-03, 1.20000000e+01],
[1.16159541e-02, 1.20000000e+01],
[8.00720396e-03, 1.20000000e+01],
[5.34312244e-03, 1.20000000e+01],
[1.11660668e-02, 1.20000000e+01],
[9.35035982e-03, 1.20000000e+01],
[9.42365925e-03, 1.20000000e+01],
[1.06079610e-02, 1.20000000e+01]])}

Table 1: Error matrix for "Clapping", "l_arm_m3"

The k-NN, or *k* Nearest Neighbors, algorithm is a frequently used algorithm in time series classification problems [6]. The k-NN algorithm works by making a final classification decision based on the top *k*

matching entries [6]. A decision criterion is created, such as an average or a mode, that takes the labels of the top *k* entries with the lowest error and makes a final prediction. The k-NN algorithm helps eliminate bias and allows for outliers to be factored in more so than a simple argmin.

Classification accuracies for each action after 300 trials for the RSM implementation are shown below in Table 2. Final decision criterion was the mode of the lowest 8 errors:

	Normal Action									
	Bowing	**Clapping**	**Handshaking**	**Hugging**	**Jumping**	**Running**	**Seating**	**Standing**	**Walking**	**Waving**
Acc %:	45.60%	95.00%	31.20%	58.00%	71.10%	52.30%	79.90%	76.50%	42.10%	15.30%
	Aggressive Action									
	Elbowing	**Frontkicking**	**Hamering**	**Headering**	**Kneeing**	**Pulling**	**Punching**	**Pushing**	**Sidekicking**	**Slapping**
Acc %:	86.00%	74.50%	39.90%	42.30%	66.00%	53.40%	83.60%	81.30%	40.00%	52.80%

Table 2: Total accuracies for the RMS transformation

Averaged Shapelet Transformations

The averaged shapelet transformation implementation is an algorithm I derived from multiple scale and shift invariance methods, mentioned in [1], and was the bulk of my focus in this study. The pseudocode for the shapelet creation aspect of the algorithm is below:

```
Initialize shapelets ← First subject listed in
training_data, testing_data ← split all data into training and testing data sets

# SHAPELET CREATION
for each action in training_data:
        for each limb in training_data[action]:
                for each subject in training_data[action][limb][1:]:
                        training_shapelet ← data series for that subject, action, limb combo
                        best_shift ← find the offset that provides the minimum error compared to the current shapelet
                        current_shapelet += training_shapelet overlap
                        current_shapelet_count += [1] where shapelet overlaps
                final_shapelet ← current_shapelet / current_shapelet_count
```

Each shapelet is initially created from the first subject in the training data list, which is randomized every trial. Every subsequent subject series in the training data list is then swept across the current shapelet using scaling and shifting methods until an ideal position is found by minimizing Euclidean distance between the two. The overlapping sections of the best fit are then added to the current shapelet, and the indexes of the overlapping sections are increased by 1 in the current_shapelet_count. After all training data series have been observed, the final shapelet for the action + limb combination is created by dividing the shapelet by its count, essentially finding the best fitting average time series sequence on the training data. Two examples of these average shapelet transformations are seen below in Figures 8 and 9:

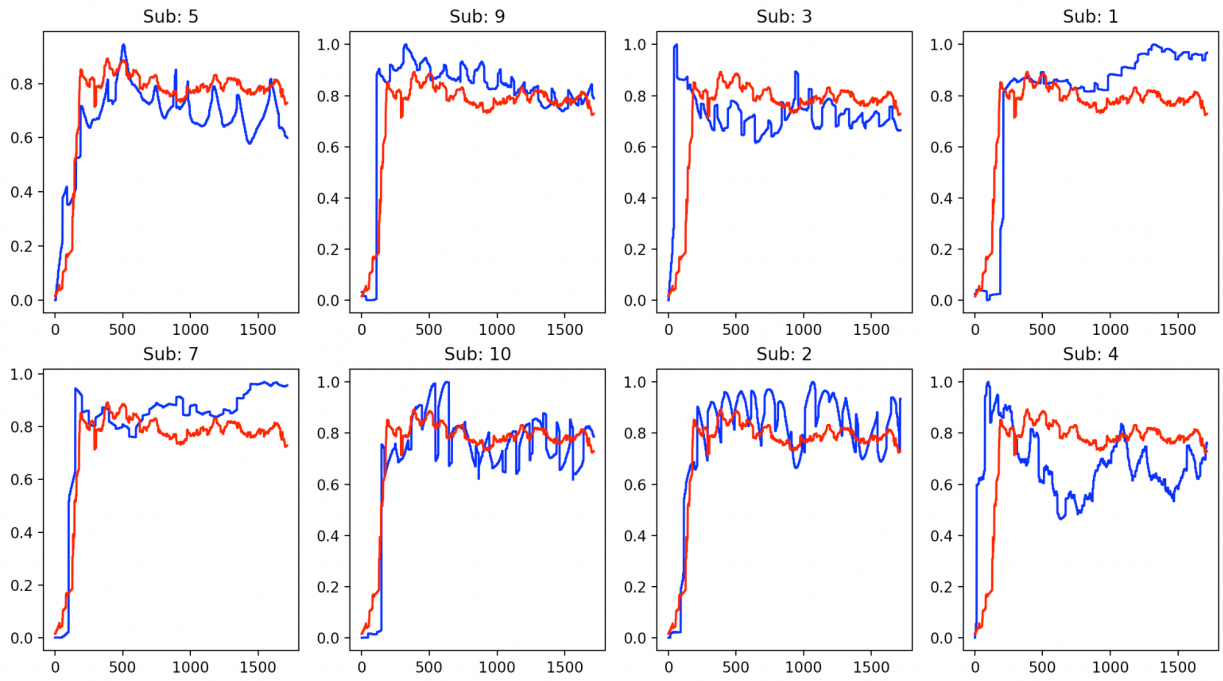

Figure 8: Average shapelet transformation (red) compared to training shapelets (blue) for action "Clapping"

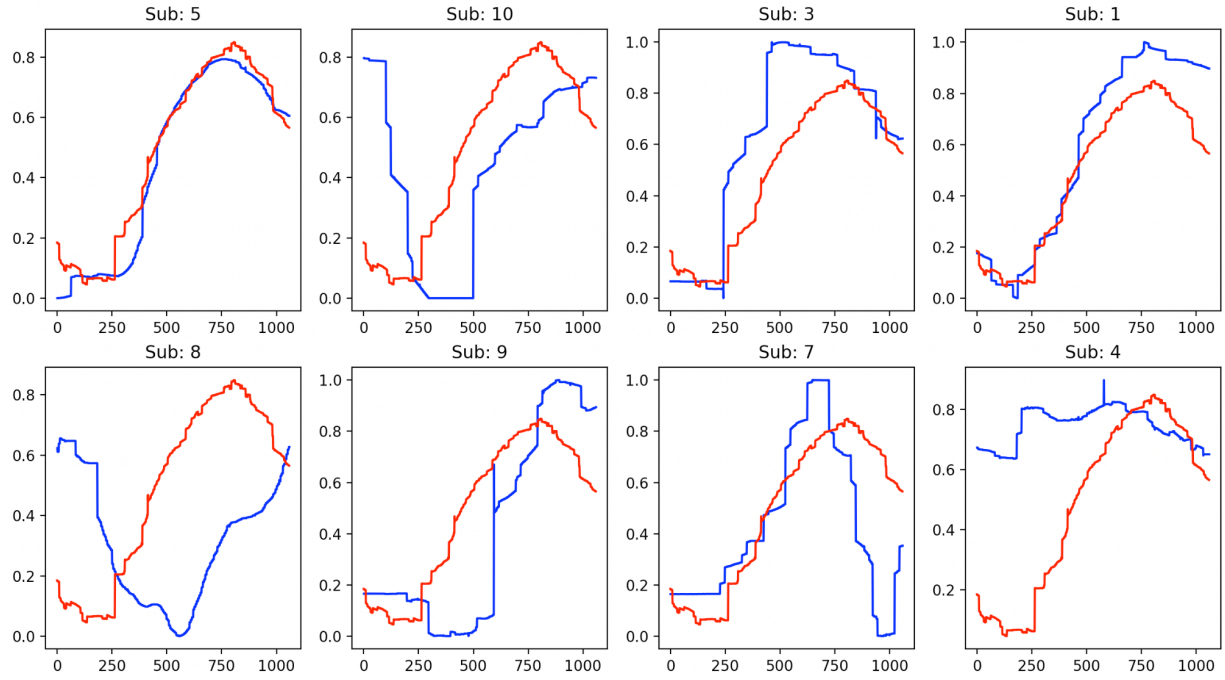

Figure 9: Average shapelet transformation (red) compared to training shapelets (blue) for action "Running"

The comparison phase of this algorithm is different from that of the RMS transformation phase. Each test shapelet is compared directly to every created action shapelet for the respective limb. The test series is given the classification of the action shapelet that minimizes the error. Results for each action after 300 trials are shown in Table 3 below:

	Normal Action									
	Bowing	Clapping	Hanshaking	Hugging	Jumping	Running	Seating	Standing	Walking	Waving
Acc %:	54.80%	93.50%	24.50%	66.00%	72.50%	65.60%	84.80%	84.00%	68.90%	11.00%
	Aggressive Action									
	Elbowing	Frontkicking	Hamering	Headering	Kneeing	Pulling	Punching	Pushing	Sidekicking	Slapping
Acc %:	97.00%	80.40%	61.50%	60.00%	81.00%	66.00%	91.50%	93.50%	59.50%	83.00%

Table 3: Classification accuracy results of averaging shapelet transformation algorithm

Sub-Shapelet Transformation

The sub-shapelet transformation is built off of the averaged transformation, where only small pieces of the created training shapelet were used in comparison. This is a common method used in practice due to typical large or continuous inputs [7]. Shapelets end up being very small subsets of the data that they represent. In this implementation, a smaller chunk of the created shapelet from the averaged algorithm is swept across each test series with no overlap. I implemented this algorithm using both the first third and middle third of the created shapelets from the averaging algorithm, obtaining better, but still poor, results for the first third, seen below in Table 4:

	Normal Action									
	Bowing	Clapping	Handshaking	Hugging	Jumping	Running	Seating	Standing	Walking	Waving
Acc %:	24.40%	61.50%	12.40%	29.50%	56.70%	34.00%	49.20%	38.50%	11.30%	4.50%
	Aggressive Action									
	Elbowing	Frontkicking	Hamering	Headering	Kneeing	Pulling	Punching	Pushing	Sidekicking	Slapping
Acc %:	23.80%	42.60%	19.90%	12.80%	42.40%	41.00%	38.70%	52.40%	38.70%	16.70%

Table 4: Classification accuracy of the sub-shapelet transformation

Results Summary

In comparing results, the averaged transformation algorithm resulted in the best accuracy on the test data after 300 trials. All of the methods follow a similar pattern in which actions performed better and which actions performed worse. For example, the action of “Clapping” performed the best in all implementations, and “Waving” seemed to perform worst. This can be explained by viewing the two actions series data:

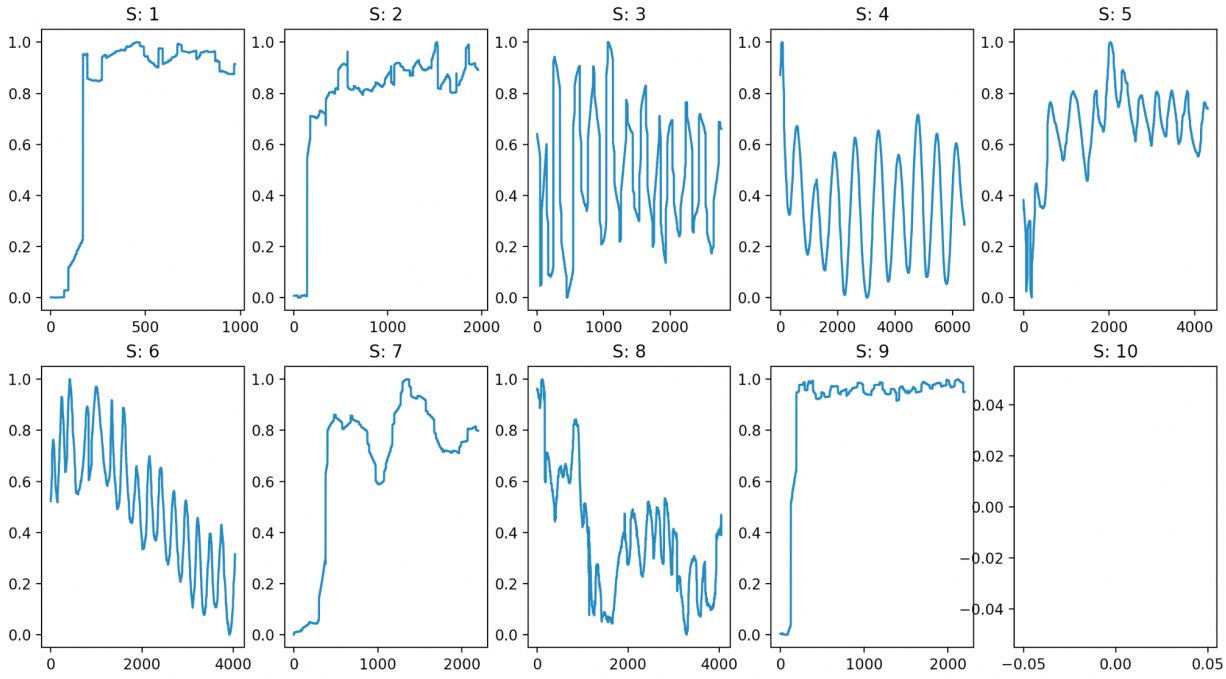

Figure 10: Normalized data for "Waving"

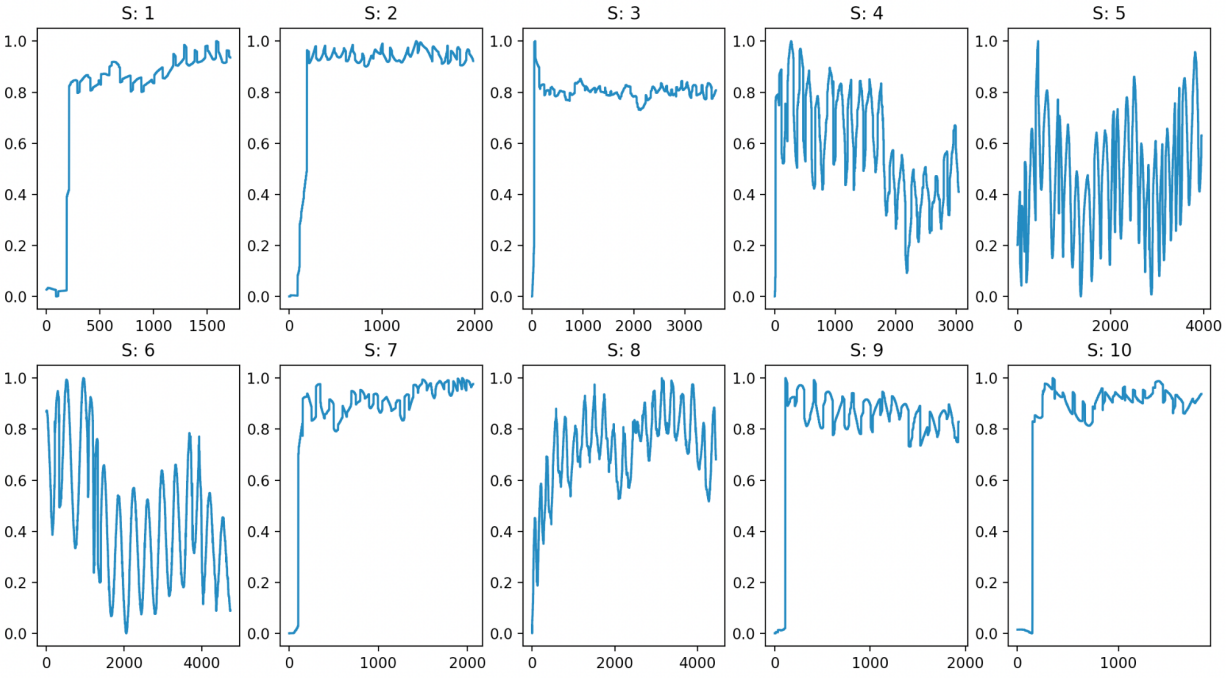

Figure 11: Normalized data for "Clapping"

Some of the patterns of the results are clear when viewing all of the data. The clapping data seems to be very representative of each other, especially in subjects 1, 2, 3, 7, 8, 9, and 10. The waving data has much

more variability in the series, with only a few subjects looking similar. This will lead to large errors in all three implementations. It is also interesting to note that some of the waving sequences look like they are in fact more representative of the clapping series. This is an issue when one of those sequences, say from subject 1, 2, or 9, gets chosen as the test data for waving in cross validation. When it gets compared to the clapping shapelet in testing, it will likely actually produce a lower error than that for its correct class, waving. This issue is likely prevalent in much of the variability in the results and is an expected outcome when attempting to classify actions so similar in motion.

Conclusion

In conclusion, I was able to develop a time series classification algorithm that outperformed typical approaches when faced with many difficulties due to data quality. The averaged shapelet transformation algorithm generalized well with training data and proved to be stable when compared to test data. The main strength to this algorithm on this dataset is likely its ability to silence outlying data points on target shapelets, and therefore be less biased in comparison. The RMS transformation is much more susceptible to large increases in error due to its bias towards the training data. This bias outweighs the decision criterion implemented by using the k-NN algorithm and shows in the overall results. In the averaged transformation algorithm, the shifting, scaling and averaging of the shapelets greatly reduces this bias and causes the shapelets to be more maximally representative of their respective class.

The relatively large amounts of error in the results can be attributed to multiple things. First, the data set used is small. There are only 10 subjects involved in the study, a very small representation of all human motion. As training data sets get bigger, shapelets created from transformation methods will become more representative of a larger group, and therefore will be able to generalize better. Second, the data is very noisy and subject motions are often not as representative of each other as necessary to implement effective time series classification algorithms. In order to obtain good classification results, it is necessary that data of the same class is consistent with itself. Some actions were much more consistent with each other than others, which is highly represented in the results of each of the algorithms presented. Lastly, known limitations when trying to solve difficult classification problems such as this one arose. Most importantly, attempting to classify actions so similar in motion. This is a large limitation in the field of time series classification and is the reason that so many class domains are kept small when facing these problems.

There are two major components to future work in this study: the first is within the implementation, and the second is within the problem space. I would love to continue to improve my shapelet search algorithm to be more shape invariant. I found it to be a difficult task to both translate and scale shapelets during the training phase and ultimately believe this area could be greatly improved. Next,

I would spend time changing the problem space to see how much results could be improved. Possible solutions include using a smaller domain, simply classifying normal vs. aggressive actions, and obtaining a larger data set. I believe the large domain space and small data set size ultimately negatively affected my results.

Individual Tasks

As I did this project alone, this section boils down to everything! My implementation consists of three main Python classes: *PreProcess*, *Visualize*, and *Analyze*. Each class is briefly described below:

The *PreProcess* class was used to do just that: preprocess the data. The main method that describes the overall process is the execute() method. Raw data was first read to a Pandas DataFrame, where custom columns of "subject", "action_type", "action" and "timestep" were added. The DataFrame was then checked for false data points and resolved those false readings via the replacing zeros process described in the Methods section. Data was then normalized via the normalization process described above and saved as a csv to be quickly read by the other classes.

The *Visualize* class was created to make multiple visuals of the data and help overall debugging during testing. Some of the helpful methods in this class include: plot_subacts(), which plots the action + limb combo for all subjects; plot_2D_both(), which plots the normalized and unnormalized data used for early debugging of the preprocessing process; and plot_offsets(), which plots the shapelet scaling and shifting transformations. The majority of the figures throughout this report are created by this class.

The *Analyze* class is where the majority of the work is done. This includes the cross validation process, the different shapelet creation and comparison methods, calculating and storing error calculations, and calculating final results and accuracies. There is a lot there, so please reach out with any questions regarding my implementation!

References

[1] L. Ye & E. Keogh. Time series shapelets: a new primitive for data mining. SIGKDD 2009. https://tslearn.readthedocs.io/en/stable/user_guide/shapelets.html

[2] Lexiang Ye. Eamonn Keogh. University of California, Riverside. Time Series Shapelets: A New Primitive for Data Mining. https://www.cs.ucr.edu/~eamonn/shaplet.pdf

[3] Timothy R. Dinger, Yuan-chi Chang, Raju Pavuluri, Shankar Subramanian. January 26, 2022. Time Series Classification. https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/

[4] T. Theodoridis and H. Hu, Classifying Aggressive Actions of 3D Human Models Using Dynamic ANNs for Mobile Robot Surveillance, IEEE International Conference on Robotics and Biomimetics (Robio-2007), Dec. 15-18, 2007, pp. 371-376.

[5] T. Theodoridis, A. Agapitos, H. Hu, and S. M. Lucas, A QA-TSK Fuzzy Model versus Evolutionary Decision Trees Towards Nonlinear Action Pattern Recognition, IEEE International Conference in Information and Automation (ICIA-2010), June 20-23, 2010, pp. 1813-1818.

[6] Theodoridis, Theo. School of Computer Science and Electronic Engineering; University of Essex. https://archive.ics.uci.edu/ml/datasets/Vicon+Physical+Action+Data+Set

[7] Gustavo E.A.P.A. Batista. Xiaoyue Wang. Eamonn J. Keogh. University of California, Riverside. A Complexity-Invariant Distance Measure for Time Series.

[8] Yen-Hsien, Lee. Chih-Ping, Wei. Tsang-Hsiang, Cheng. Ching-Ting, Yang. 10 January 2010. National Taiwan University, Taipei. https://www.sciencedirect.com/science/article/pii/S0167923612000097

[9]H. Harry Asada. Introduction to Robotics. Massachusetts Institute of Technology Department of Mechanical Engineering.https://www.studocu.com/row/document/jamaa%D8%A9-aayn-shms/industrial-robotics/chapter-3-robot-mechanisms/8114159

[10] Rutuja Pawar. 15 June 2021. k-NN based Time Series Classification. https://towardsdatascience.com/k-nn-based-time-series-classification-e5d761d01ea2